NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A DEVELOPMENT AND SIMULATION OF
SYNERGISTICALLY INTEGRATED RELIABILITY (SIR)
FOR AN ULTRA-RELIABLE FAULT TOLERANCE
COMPUTER UNDER COMMUNICATION SOFTWARE
PROTOCOL FOR THE GROWTH ALGORITHM

by

Nophadol Sudhamasapa

September 1986

Thesis Advisor: Larry W. Abbott

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) 62	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) A DEVELOPMENT AND SIMULATION OF SYNERGISTICALLY INTEGRATED RELIABILITY (SIR) FOR AN ULTRA-RELIABLE FAULT TOLERANCE COMPUTER UNDER COMMUNICATION SOFTWARE PROTOCOL FOR THE GROWTH ALGORITHM (UNCLASSIFIED)

12 PERSONAL AUTHOR(S)
SUDHAMASAPA, Nophadol

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1986 September	112

16 SUPPLEMENTARY NOTATION

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Fault Tolerant Computing, Distributed I/O, Ultrareliable Systems

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis describes a C language implementation of a GROWTH algorithm to manage a fault-tolerant computer communication network. Simulation tools are developed to verify and validate the operation of the GROWTH algorithm, and concepts are developed to evaluate the performance of the communication network.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Larry W. Abbott	22b TELEPHONE (Include Area Code) 408-646-2379 / 22c OFFICE SYMBOL 62At

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

Approved for Public Release; Distribution is Unlimited

A Development and Simulation of Synergistically Integrated Reliability (SIR) for an Ultra-reliable Fault Tolerance Computer under communication software protocol for the GROWTH ALGORITHM

by

Nophadol Sudhamasapa
Lieutenant, Royal Thai Navy
B.S.E.E., Royal Thai Naval Academy, 1979

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from

NAVAL POSTGRADUATE SCHOOL
September 1986

ABSTRACT

This thesis describes a C language implementation of a GROWTH ALGORITHM to manage a fault-tolerant computer communication network. Simulation tools are developed to verify and validate the operation of the GROWTH ALGORITHM, and concepts are developed to evaluate the performance of the communication network.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION AND OVERVIEW

The exceptionally fast growth in the area of high performance Very Large Scale Integrated Circuits (VLSI) is rapidly reducing the cost of processors while providing increased processing power to microprocessors. Low cost microprocessor chips are quickly reducing the price of fault-tolerant computer architectures of the recent past. Although technical difficulties still exist, the massive redundancy required for fault tolerant designs is no longer prohibitive in cost. It is now possible to achieve the requisite reliability with relatively inexpensive fault tolerant computer hardware. Because in many cases reliability is the critical system constraint, complex computers with higher reliability are being demanded for a growing number of vastly different applications.

Digital computing systems are being used in most industries today, including aviation, space and industrial control. Systems which are very difficult to service, such as satellites, or systems critical to the safe operation of a plant are dependent on the reliable operation of their computer driven controller. If the computer is unreliable in its operation, catastrophic disaster may occur. Figure 1 [Ref. 1] shows the general effects of a fault in a digital system. The causes of hardware or software faults can be

categorized into four classes:

1. Specification errors.
2. Implementation errors.
3. External disturbance.
4. Random component failure.

The resultant effect of the above faults is a system malfunction. Figure 2 [Ref. 1] shows the boundary lines obtained through the use of fault tolerant techniques in order to maintain a system's normal performance. These techniques are called: fault avoidance, fault masking and fault tolerance. [Refs. 1, 2, 3]

Fault avoidance (fault intolerance or fault prevention) attempts to prevent faults from ever occurring by screening out sources of faults. Techniques for fault avoidance include thorough design reviews, screening of components prior to installation, and testing (system validation) the completed system. Using fault avoidance, the probability of a system failure can be reduced to an acceptably low value.

Fault masking eliminates the presence of faults while providing continuous system operation. One example of fault masking is a triple modular redundant system with a voting scheme.

Fault tolerance tries to produce correct results in the presence of errors. A fault tolerant system accomplishes this by eliminating the cause of an unreliability, which is a

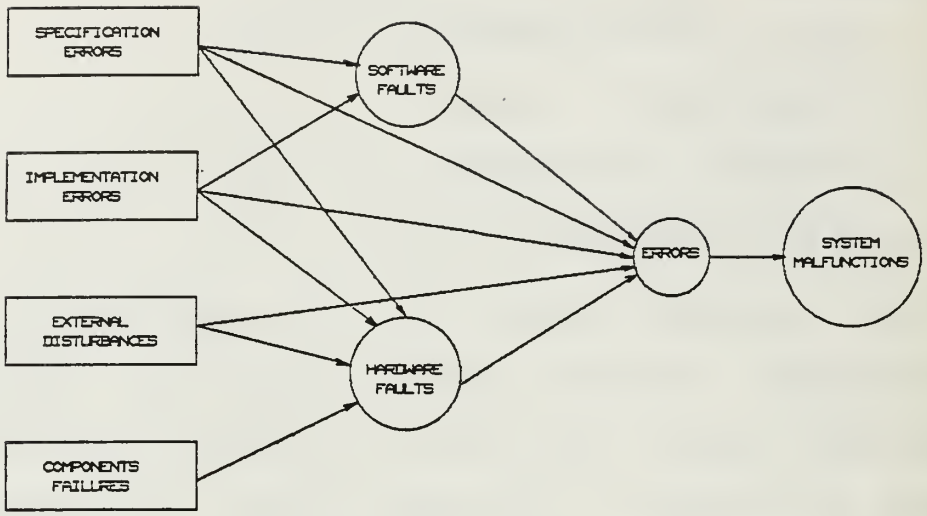


Figure 1 Cause-Effect Relationship of Faults [Ref. 1]

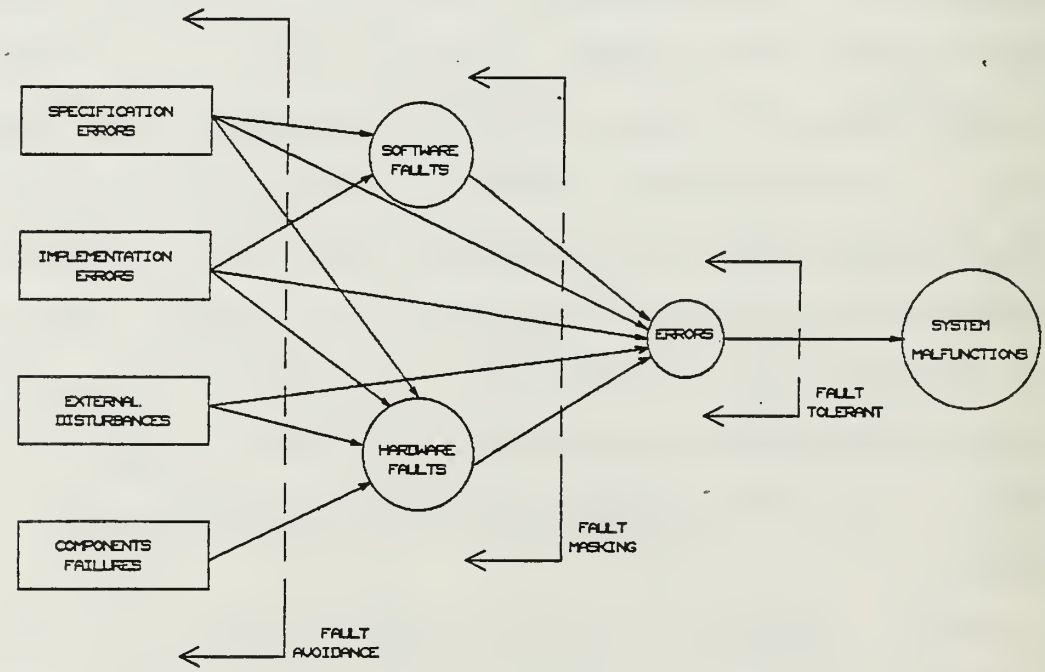


Figure 2 Barriers Constructed by Fault Avoidance, Fault Masking and Fault Tolerance [Ref. 1]

necessary step if applications requiring a high degree of reliability are to be supported. For example, one fault tolerance approach recognizes that hardware and software faults are possible and negates the effects of errors through the use of protective redundancy.

There are three forms of protective redundancy:

1. Hardware redundancy.
2. Software redundancy.
3. Time redundancy (repetition of operations). [Refs. 1, 2, 3, 4]

Hardware redundancy includes additional hardware components and can be adopted as either static or dynamic hardware redundancy. Static hardware redundancy ensures that a failure has no effect on the output and hence is masked from the actual operation through the additional of redundant hardware components. A common method of obtaining static redundancy is through triple modular redundancy with voting. Dynamic hardware redundancy, allows faults to occur and propagate until detected. Once detected the appropriate recovery action (self-repairing computer system) is activated.

Software redundancy includes additional programs, program segments, and microprogram steps which are mainly used as a supplement to hardware redundancy. Software redundancy can also be used as a diagnostic program which insures faults do not exist in a fault-free computer.

Time redundancy consists of repeating the execution of a module and checking results. The detection and correction of errors caused by temporary faults can be corrected. However, time redundancy only works for transient faults through the use of time redundancy and is not suitable for permanent faults.

In response to a failure a fault-tolerant system performs four steps. These steps are fault detection, fault location (diagnosis), fault containment (isolation) and fault recovery (correction). Systems which require a high degree of availability only need a fault detection scheme because diagnosis and correction can be done off line. For online ultra-reliable systems, diagnosis and correction must also be done automatically. [Refs. 1, 2]

Fault detection determines if a fault has occurred in a system. Fault detection is required prior to implementing any recovery mechanism. Fault detection mechanisms can be employed both in local computers and in intercommunications between computers within a network. With a single computer, fault detection is performed with special hardware which detects faults within the computer electronics. In a network system, fault detection involves running the same program on two or more computers, and comparing results through use of voting to check a computer's output.

Fault location determines specifically where a fault has occurred and then activates an appropriate mechanism.

Fault recovery is the correction of a fault which maintains a system's reliability.

Basic definitions:

1. VOTED RECOVERY determines a correct result by a majority vote of the outputs from several modules. If disagreeing modules can be replaced with spares (under control of agreeing modules), this self-healing form of voted recovery is termed HYBRID REDUNDANCY.

2. HARDCORE ITEMS are circuit elements whose failure could disable the complete computer network or large portions of it. Hardcore items require thorough identification and careful protection against faults which are common in the communications network or fault recovery mechanisms.

3. INTERSTAGE is the buffer storage control. The interstage provides full duplex data transfer for the direct link between the CPU and the external network. The interstage stores data from external network for the voter and the CPU, and the interstage can recirculate the stored data to provide data congruency and transmits to the external interstage.

A. SYNERGISTIC FAULT-TOLERANT COMPUTER

In order to achieve the requisite degree of performance for an ultra-reliable, fault-tolerant computer system, a synergistic combination of hardware and software fault-tolerant techniques must be combined. A number of state-of-the-art fault-tolerant techniques are integrated to form a single, unified concept of fault-tolerance. These techniques include: hybrid redundancy, N-version programming, interstages, and a reconfigurable, redundant I/O network. A combination of hybrid redundancy, N-version programming, congruent data interchanges, and hybrid redundancy management (redundancy management achieved by the collaboration of

hardware and software) define Synergistically Integrated Reliability (SIR). SIR is a simultaneous combination of both hardware and software fault tolerant techniques which achieves a high degree of reliability.

The key system elements for a candidate fault-tolerant computer architecture are depicted in Figure 3. The system will achieve the broad architectural goals for fault-tolerant systems. The sensors, effectors, and the computation elements are integrated to form an ultra-reliable, integrated digital system through the use of an ultra-reliable data communication systems, such as the Dispersed Sensor Processing Mesh (DSPM) [Ref. 5].

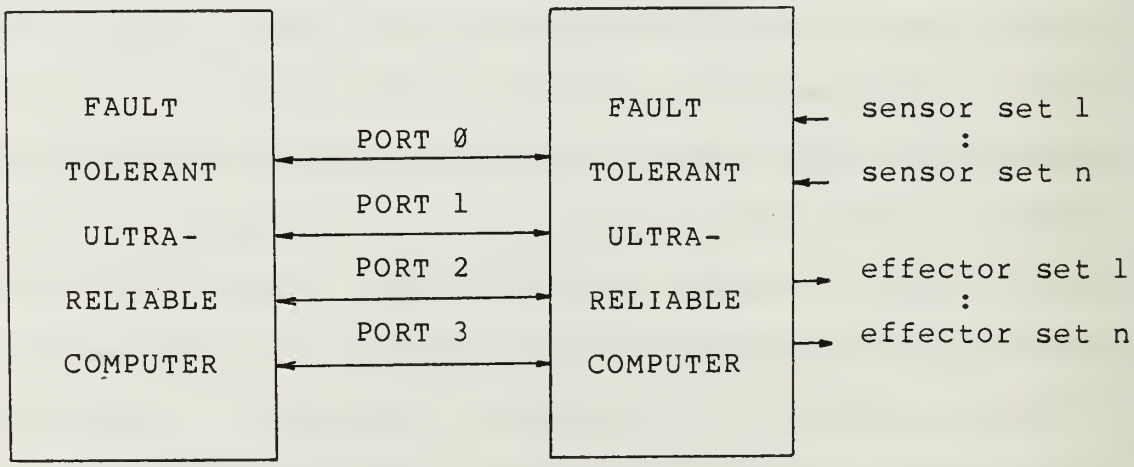


Figure 3 Key Hardware Element of the Candidate Fault-Tolerant Computer

Ultra-reliable systems achieve their reliability by designing systems with replicated components. N-Modular Redundancy (NMR) is used when whole hardware systems are to be replicated. One technique of NMR, hybrid redundancy, combines NMR and standby sparing to increase system reliability. In light of the over-all results from the synergistic, fault-tolerant computer experiment, hybrid redundancy was chosen as the most likely candidate for implementing an ultra-reliable computer.

In a triplex core hybrid redundancy system(see Figure 4), three processors (P_i) are selected to run the same computer problem by the voter. If the results are not unanimous, a feedback signal is sent to the rotary multiplexer with the identity of the faulty computer. The multiplexer then

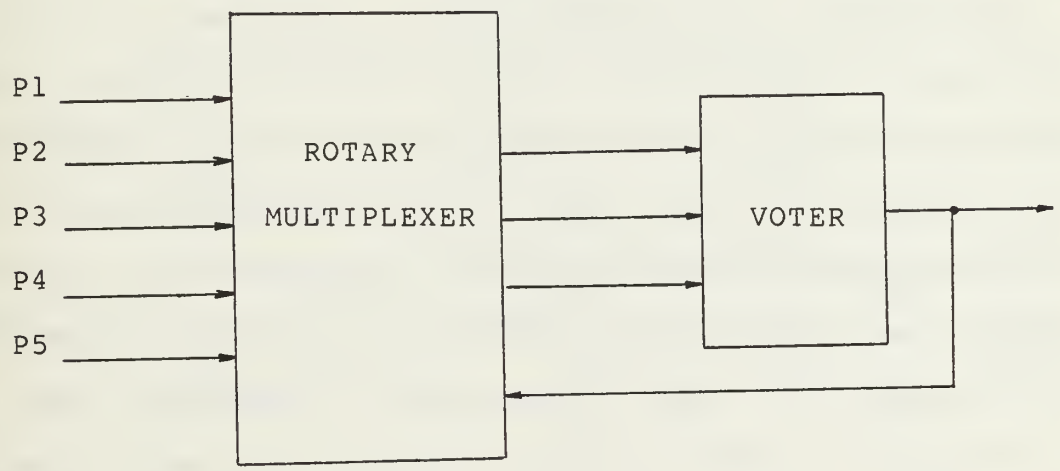


Figure 4 Basic Hybrid Redundancy

deactivates the fault computer while activating the highest priority spare on the rotary multiplexer. In addition to swapping processors, the rotary multiplexer can also retry a failed processor at a later time to see if the failure was caused by a transient fault.

In the hybrid redundancy architecture the voter and multiplexer form the hardcore, or that part of the circuit which must function for successful intercommunication between computers.

Each node in the SIR architecture (Figures 5 and 6), is composed of the following elements: a computer, interstage, voter, and rotary multiplexer. All of these components are necessary to implement hybrid redundancy, N-version programming, congruent data interchanges, and hybrid redundancy management. Figure 6 shows the block diagram implementation of a computer and its hardcore.

N-version programming is a software reliability technique. In circumvents generic software faults through the use of several processors running different versions of a program but derived from the same software specification. The differences between software languages will prevent a similar fault from simultaneously occurring in all versions.

Another of the synergistic techniques used, the interstages, assures source congruency for data transferred between channels. The interstages also provide a means for reliable transfer between channels when more sophisticated

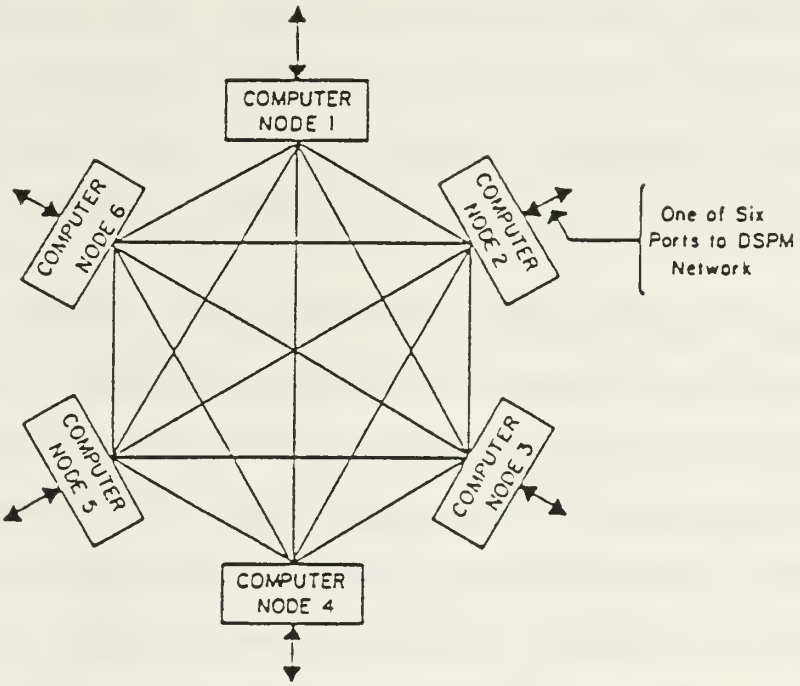


Figure 5 Overview of SIR Architecture

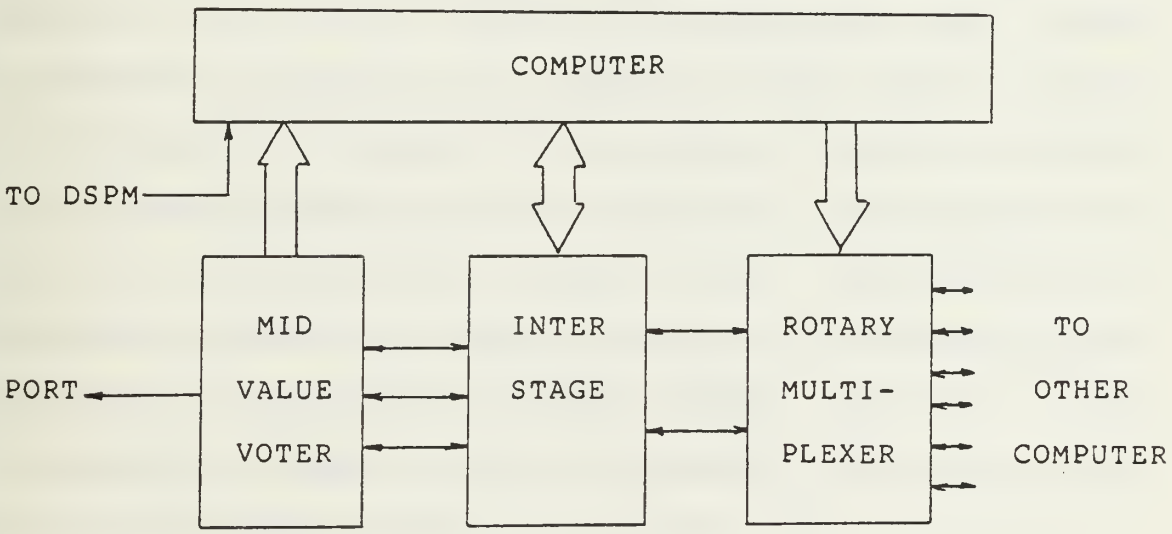


Figure 6 Simplified Node of SIR Architecture

redundancy management is required. The required interaction between the computer redundancy management software and the hardcore suggests the use of an intercommunication network between each hardcore (see Figures 5 and 6).

B. DISPERSED SENSOR PROCESSING MESH (DSPM)

Fault tolerant systems require an ultra-reliable communications structures to operate with the ultra-reliable computational element previously described. The Dispersed Sensor Processing Mesh (DSPM) is an ultra-reliable structure used in fault tolerant computer communications networks. DSPM exhibits an organic-like ability to regenerate itself after suffering damage.

A strategy to implement the Dispersed Sensor Processing Mesh (DSPM) was suggested by Thomas B. Smith III [Ref. 5]. Smith uses dedicated transmission paths on a dynamic, reconfigurable nodal network to incorporate sensors and effectors into an ultra-reliable integrated digital flight control system. The DSPM system provides the same level of ultrareliability for data communication that the Fault-Tolerant Multiprocessor (FTMP) or Software Implemented Fault Tolerance (SIFT) systems provide for the computation unit. The DSPM through use of a complex communication architecture limits damage propagation by prompting system adaptability. The primary goal of DSPM is to overcome deficiencies of conventional bused data communication systems

currently in use. In review a SIR through use of the DSPM is used for gathering sensor data and distributing effector data into an ultra-reliable nodal network. The SIR architecture acts as a central bus and controller performs algorithms to grow and maintain the network.

Figure 7 [Ref. 5] depicts a typical DSPM network comprised of a central bus controller (BC), simplex nodes, and communication links (point to point MIL-STD-1553 variant). Due to the central location of the bus controller in the DSPM architecture, it must be fault-tolerant. To emphasize the requirement for a fault-tolerant bus controller, the bus controller is shown as a redundant, four channel (quadraplex) computer. In DSPM, all networking intelligence is located in the bus controller. In particular, the bus controller contains the growth algorithm used for growth of initial communication trees to other nodes. In addition the bus controller initiates and controls all communication traffic on the network. The bus controller has four communication ports, one for each channel. The ports are numbered in a clockwise order. The bus controller is programmed with three main routines: the GROWTH ALGORITHM, the REPAIR ALGORITHM, and the MODIFY ALGORITHM. These determine the tree structures which will interconnect nodes for data flow and selection of active links.

The GROWTH ALGORITHM is used to create an initial network configuration which avoids existing network faults. Once the

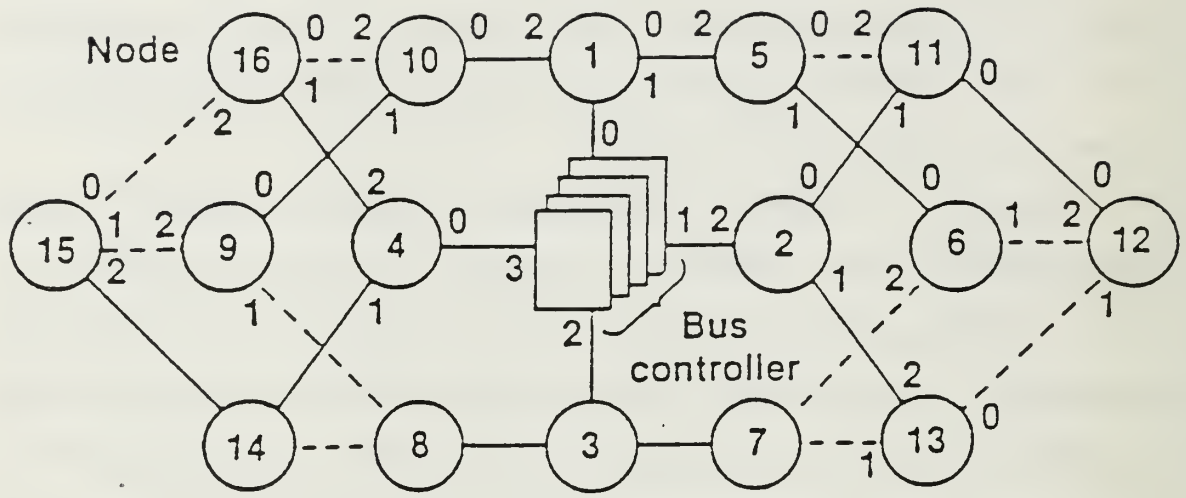


Figure 7 DSPM16 Generic DSPM-Type Network

initial network configuration is grown, the REPAIR ALGORITHM is used to reconfigure the network as each additional fault is detected. If concurrent multiple faults occur, the computer program will revert to the GROWTH ALGORITHM and regrow the entire network.

Nodes are non-redundant, simplex computers which gather data and drive actuators under the guidance of the bus controller. The nodes are as simple as possible and lack significant intelligence. Each node in Figure 7 except the bus controller has three communication ports, which connect node to the network. The communication links which numbered clockwise, are full duplex, point to point, transmission

paths. In the DSPM approach, all links in the network can not be active at the same time. Only the active links (see Figure 7 shown as solid lines) actually carry information. The inactive links (see Figure 7 shown as dotted lines) are unused links and do not carry any information.

The DSPM system is formed by a cyclic nodal network, but the active communication network actually forms a tree of active links coming from the bus controller. The communication paths in a DSPM network are formed by tree structures of active links starting at the bus controller and growing to the furthest leaf node. In Figure 7 four trees are shown. One tree which is typical of the rest, starts at bus controller port 3 and goes to node 4 port Ø. This tree contains nodes bus controller, node 4, 14, 15 and 16. Node 15 is the successor of node 14 and node 14 is the predecessor of node 15. Nodes 15 and 16 are the leaf nodes and bus controller port 3 is the root.

After a network is grown by the GROWTH algorithm, the DSPM system operates in two ways to handle the additional failures through the use of the REPAIR algorithm and MODIFY algorithm.

A system will try to repair itself when a failure is detected by calling the REPAIR ALGORITHM routine. When in the REPAIR program another failure is encountered, REPAIR is aborted and the network is regrown by the GROWTH algorithm routine. Allowance for only one failure before rebooting the

growth program, is purposely done to keep the REPAIR algorithm as simple as possible. This feature is important because a REPAIR may only be partially completed when another fault is detected. [Ref. 6]

The MODIFY ALGORITHM is designed to eliminate unobservable faults on inactive links of the communication network. It is possible grow an entire network without communicating through all links. A DSPM may contain undiscovered failures which are called latent faults. These faults will remain hidden within the network until an attempt is made to activate the specific faulty link. The MODIFY algorithm resolves the problem of hidden faults by continuously replacing active links with inactive link. The true status, fault tolerance and current condition of a specific DSPM system will not be known unless the MODIFY algorithm is included. [Ref. 6]

An example of a specific DSPM communication system, which is interfaced to NASA's F-8 Digital Fly-By-Wire (DFBW) "IRONBIRD" simulator [Ref. 7]. As shown in Figure 8, the IRONBIRD simulator provides a safe yet realistic means of testing the complex interaction of a highly reliable communication network with a state-of-the-art, triply redundant, digital flight-control system. The flight control system contains redundant computers, sensors, actuators and flight-critical software. The DSPM experimental system consists of three major components: The F-8 DFBW Ironbird

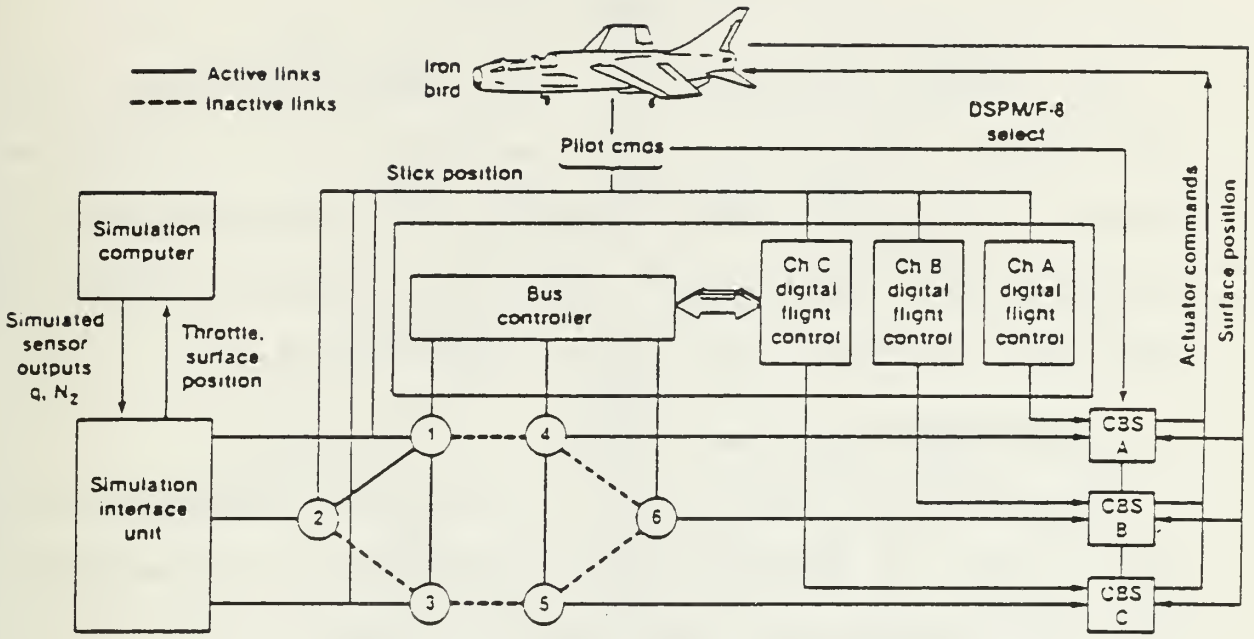


Figure 8 DSPM Experimental System

simulator, with triplex flight-control computer and control-law software; a central computer with simulation software; and a six-node (plus a triplex bus controller) version of the DSPM network [Ref. 8].

The specific goals of this thesis are:

1. To develop the software for the GROWTH ALGORITHM, using standard high level C-Language on an IBM_AT microcomputer.

2. To develop a concept for DSPM communication network performance evaluation.

II. GROWTH ALGORITHM

The GROWTH ALGORITHM is the computer program which initializes the growth of tree structure for a Dispersed Sensor Processing (DSPM) network. The GROWTH algorithm is controlled and operated from the bus controller which is physically located in the center of the network.

A. PROCEDURE FOR GROWTH

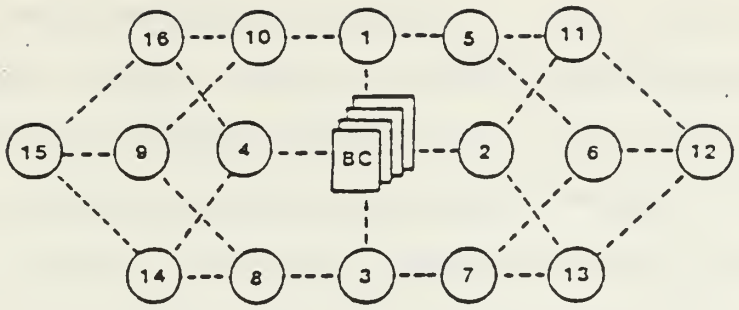
There are two instances when the GROWTH algorithm will be used: first on initialization of the network, and second if there are multiple concurrent faults detected.

Each of the above cases will be individually addressed with an example given on how growth actually occurs.

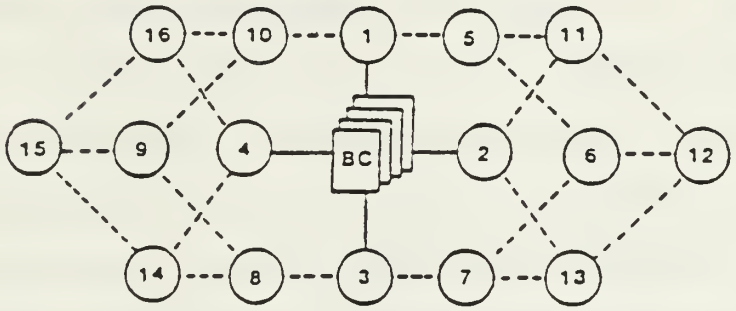
1. Initialization

The bus controller provides the stimulus for growth to occur. Growth is directed outward from the bus controller to form structures called trees with neighboring nodes. The first step in forming a network is to connect all nodes with inactive communication links (see Figure 9a).

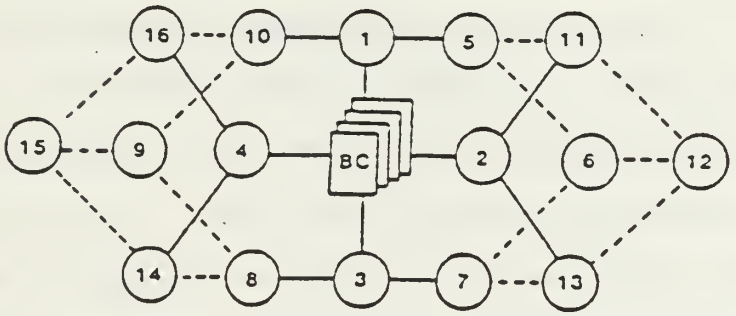
Before allowing a communication link to become active, which effectively grows a node onto the network, two criteria must be met. First, the node to be added must not already be a member of the network. That is, the node must not be connected with an active link. Second, the candidate node must return a proper status word, which defines all



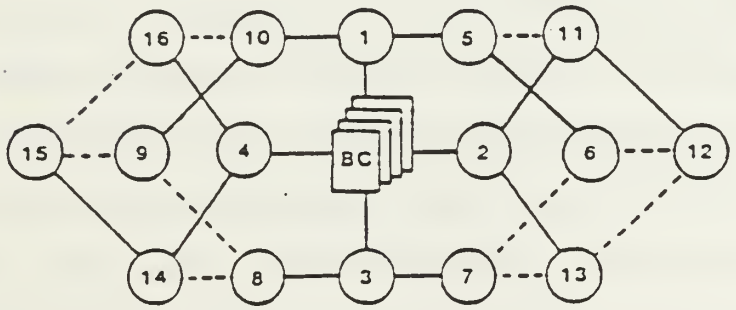
a. Unformatted (bus controller)



b. Growth Cycle 1 (1,2,3,4)



c. Growth Cycle 2 (5,10,11,13,7,8,14,16)



d. Growth Cycle 3 (6,9,12,15)

Figure 9 An Example of Initialization Growth

states in the node, to the bus controller when instructed to activate its INBOUND port. If the proper status word is returned the link and node are deemed to be faultless. An improperly returned status word implies a fault exists in the link and/or node, and causes the link to remain inactive.

The bus controller is obviously the starting point for growth. If the first neighboring node, designated node 1, queried, responds with a correct status word, it is connected to the bus controller from port Ø. Once the connection is made and checked, the link and node are put in the First- In-First-Out (FIFO) table. Growth cycle continues with the bus controller ports 1, 2 and 3 and nodes 2, 3 and 4 (see Figure 9b). Once the root node (bus controller) has linked to a node on all four ports, reference to the bus controller is removed from the FIFO leaving only the nodes comprising the next level in the tree. As an example in Figure 9b, when the first cycle is completed the FIFO would contain (1, 2, 3, 4).

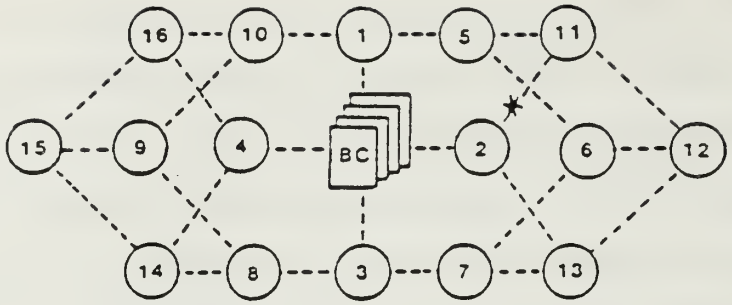
The next growth cycle (Figure 9c) starts with the leading entry in the FIFO (node 1). Node 1 then becomes the point where growth cycle 2 starts its queries to neighboring nodes. Neighboring nodes which are not already members of the network will be linked to node 1, in this case it is nodes 5 and 1Ø. Once growth in node 1 is completed the FIFO contains (2, 3, 4, 5, 1Ø). Node 2 is now pulled from the queue and the GROWTH algorithm queries the neighboring nodes

of node 2. This process, assuming no faults are detected, ends with nodes 11 and 13 being added to the network, and the FIFO which now contains (3, 4, 5, 10, 11, 13). Growth cycle 2 continues in this manner until nodes 3 and 4 have been processed, at the end of cycle 2 the FIFO contains (5, 10, 11, 13, 7, 8, 14, 16).

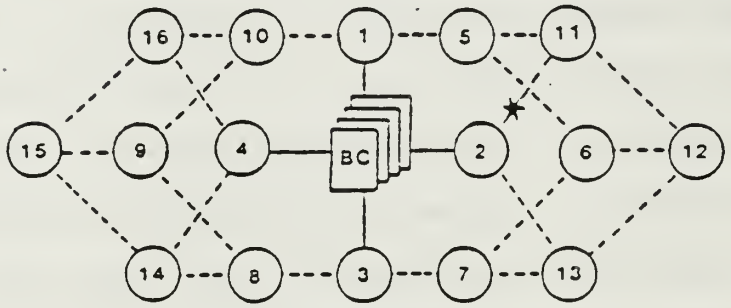
The third growth cycle (Figure 9d) continues the process described above until all neighboring nodes in the network are members. In this example cycle 3 ends the growth process because all nodes are now network members. If there were more nodes the growth cycles would continue until all nodes were grown into the network.

2. Growth Process when Faults are Present

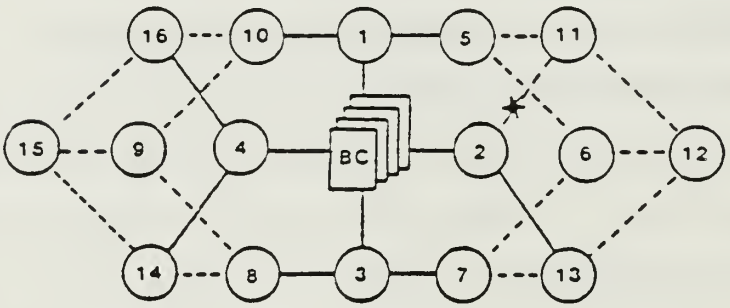
The GROWTH algorithm is also activated when a second fault is detected prior to the REPAIR algorithm completing on a previous fault. Figure 10 shows an example of how growth occurs in the presence of a fault. The fault is depicted as a star on the line between nodes 2 and 11. The growth process proceeds as described in the initialization process until it reaches growth cycle 1, node 2. At this point the algorithm recognizes that the link between nodes 2 and 11 is failed and does not attempt to grow node 11 to 2. The growth process then proceeds again as before until growth 3, node 5. At this point node 5 has grown to node 11 as well as node 6. The important concept in the GROWTH algorithm is that an entire network of nodes can be generated without



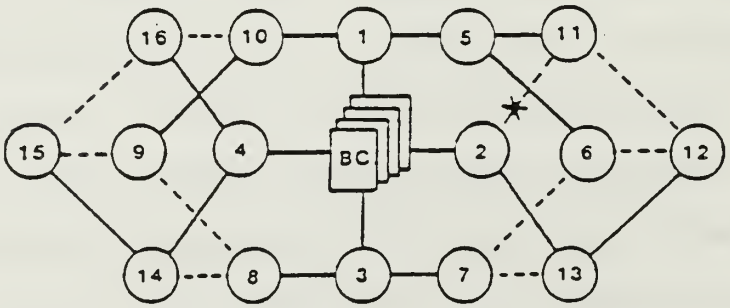
a. Unformatted (bus controller)



b. Growth Cycle 1 (1,2,3,4)



c. Growth Cycle 2 (5,10,13,7,8,14,16)



d. Growth Cycle 3 (11,6,9,12,15)

Figure 10 An Example of Multiple Fault Growth

communicating on all links. However, there also exists the possibility of hidden faults residing in the network. All dashed lines in Figure 10d could potentially contain faults which would be unidentified until an attempt to activate the link were made.

B. FLOWCHART

Figure 11 shows a simplified flowchart of the GROWTH algorithm. The bus controller software must initialize a

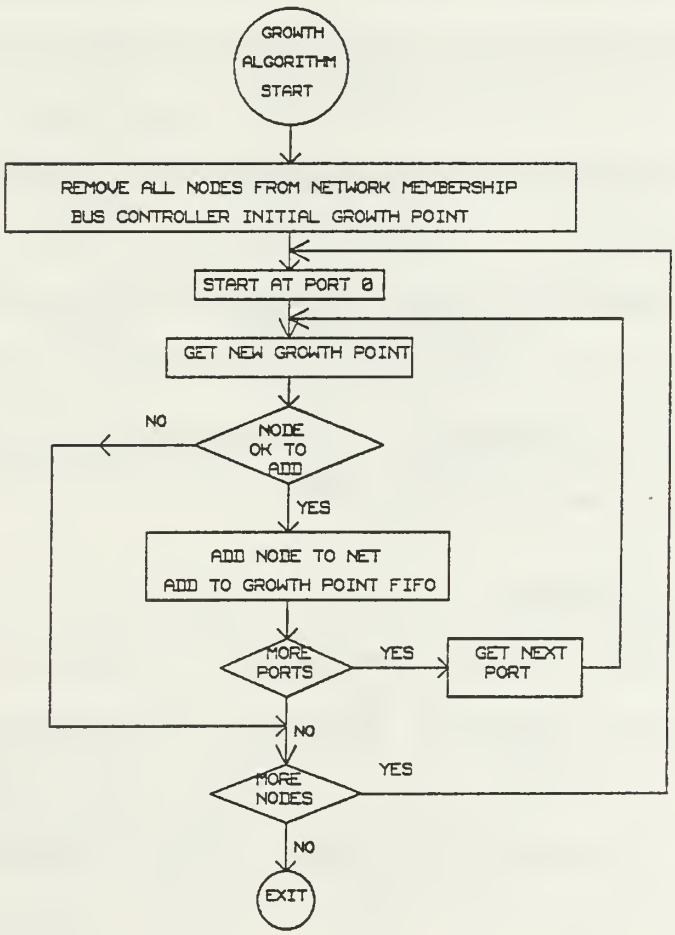


Figure 11 Simplified Growth Algorithm flowchart

LINK_STATUS table prior to the start of the growth process. The LINK_STATUS table is initialized to indicate that each link is fault free but not TESTED. Locations of nonexistent points are initialized as USELESS links in the link status table.

The first step in the flowchart initializes the data structure (remove all nodes from network membership) and sets the GROWTH_FIFO to "Ø"(make the bus controller the initial growth point). The next step of the growth process starts with bus controller port Ø. If the destination node accepts the connection, the bus controller inserts the destination node into the growth point table (GROWTH_FIFO) and activates the destination node's INBOUND port. The data structure is then manipulated and the bus controller port 1 is selected for the next iteration. After all the ports on the current growth points are processed, the GROWTH algorithm goes to the next node for processing. This sequence continues until all nodes in the network processed.

C. DATA STRUCTURE

The GROWTH algorithm requires a data structure to store information on how the network is configured. The data structure necessary to implement the GROWTH algorithm is strongly influenced by the job of tracking the growth process. Figure 12 shows a sample source code for a data structure used in the growth algorithm.

```
/*************************************************************/
/*      DEFINE DATA STRUCTURE FOR GROWTH PROCEDURE           */
/*************************************************************/

                #include <stdio.h>

                #define  INACTIVE     0
                #define  INBOUND      1
                #define  OUTBOUND    -1
                #define  USELESS      2
                #define  FAILED       1
                #define  MEMBER       1
                #define  NOT_CONN    -1
                #define  NOT_A_MEMBER 0
                #define  BUS_CONT     0
                #define  NUMNODES NNODES+1
                #define  NUMPORTS NPORTS+1

        /* DEFINE CONNECTION ARRAY 2 DIMENSIONS */
                int      NODE_CNN [33] [7];
                int      PORT_CNN [33] [7];

        /* DEFINE STATUS ARRAY 2 DIMENSIONS */
                int      PORT_STATUS [33] [7];
                int      LINK_STATUS [33] [7];

        /* DEFINE NETWORK TABLE ARRAY 1 DIMENSIONS */
                int      GROWTH_FIFO[33];
                int      NODE_ROOT[33];
                int      NET_MEMBER[33];

        /* DEFINE VARIABLES INPUT INFORMATION*/
                int      NNODES,NPORTS;
```

Figure 12 Source Code Listing for Data Structure

The data structures used are as follows:

1. Network connectivity
   - Node connection array (NODE_CNN[][])
   - Port connection array (PORT_CNN[][])

2. Network status
   - Port status array (PORT_STATUS[][])
   - Link status array (LINK_STATUS[][])

3. Network table
   - Net member (NET_MEMBER[])
   - Growth point (GROWTH_FIFO[])
   - Node root (NODE_ROOT[])

Network connectivity is defined by two arrays, the node connection array and the port connection array. nodes are represented by matrix rows, while ports are identified by the columns of the matrix. In the node connectivity array (Figure 13a), elements in the array designate the destination node for the link coming from the source nodes port. A "-1" entry means that no destination exists for that port on the node. In a like manner, the port connection array, (Figure 13b), shows the destination port number from a particular nodes port. A "-1" entry in this array again means that the port is not connected to the network.

Network status can also be defined by two different arrays the port status array and the link status array, using the same logic structure as network connectivity.

There are three possible states for each element in the port status array: INACTIVE, INBOUND and OUTBOUND. INACTIVE status is marked by a Ø whenever a port is turn off by the bus controller. INBOUND and OUTBOUND does not imply unidirectional ports, but indicates port characteristics based on the port's position in the network. An INBOUND port is a port that relays bus traffic from the tree to the bus controller. There is only one INBOUND port per active node. An OUTBOUND port is a port(s) which relays bus traffic from

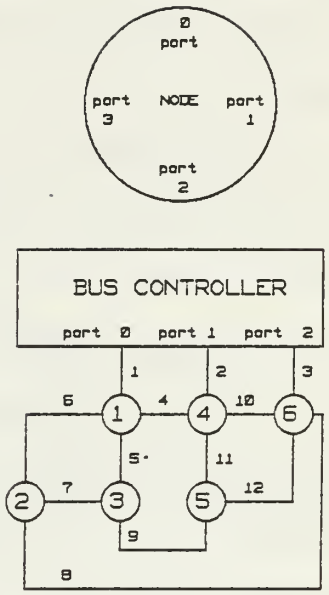
NETWORK CONNECTIVITY

NODE	PORT 0	1	2	3
0	1	4	6	-1
1	0	4	3	2
2	1	3	6	-1
3	1	-1	5	2
4	0	6	5	1
5	4	6	3	-1
6	0	2	5	4

a. NODE CONNECTION

NODE	PORT 0	1	2	3
0	0	0	0	-1
1	0	3	0	0
2	3	3	1	-1
3	2	-1	2	1
4	1	3	0	1
5	2	2	2	-1
6	2	2	1	1

b. PORT CONNECTION



c. DSPM6 STRUCTURE

Figure 13 An Example of DSPM Structure and Network Connectivity Array

the bus controller to the nodes it wants to communicate with. When the port is activated the port must be marked as INBOUND (1) or OUTBOUND(-1), at the time of activation.

The link status array contains information on how the links are connected to each port in the network. Link status array elements have three possible states: USELESS, TESTED and FAILED. Only the USELESS and FAILED status states are of

interest to the GROWTH algorithm since the TESTED state is only used in the MODIFY routine [Ref. 6]. The purpose of FAILED obviously means the link has a fault and can not be used.

The USELESS state is used to identify ports which are not connected to the network and to inhibit growth from the network back into the bus controller. The growth algorithm creates the USELESS state for a link that attempts to connect to the bus controller. The bus controller creates a FAILED state when a failure is discovered at a port. The USELESS status is redundant since the presence "-1" in node connection array indicates that no link has been connected to the port.

The network table contains information on the current membership in the DSPM network. The network table consists of net member, growth point and node root tables.

The net member table (NET_MEMBER[]) contains the information necessary to end the GROWTH algorithm once all nodes have become members of the network. If a node is not a member of the network, the node's location in the NET_MEMBER table contains Ø.

The growth point table (GROWTH_FIFO[]) contains the growth point nodes in the order they are added by the GROWTH algorithm. GROWTH_FIFO is a one dimensional array with length equal to the number of nodes (including the bus controller) in the DSPM network. The GROWTH_FIFO starts with

the bus controller (Ø). As the GROWTH algorithm proceeds, nodes which provide potential growth points are entered into the GROWTH_FIFO. The current growth point is determined by a growth pointer (head pointer). The growth pointer points to the upper most unprocessed growth point in the GROWTH_FIFO. When processing of the current growth point is completed, the growth pointer is incremented to the next location in the GROWTH_FIFO.

The node root table (NODE_ROOT[]) is generated while the GROWTH algorithm is running. It defines which port of the bus controller is the root of the tree on which a particular node is located.

D. GROWTH PROCEDURE

The growth procedure implemented in this paper is based on the GROWTH algorithm concepts previously discussed and operates in both a fault free as well as a faulty network. The main processes are designed in accordance to the simplified flowchart described in the previous section. The growth procedure is written in standard high level C-language to maintain a portable software module for use on different computer systems and/or different operating systems with a minimum amount of modifications. Figure 14, 15, 16 show the listings of the growth procedure.

The growth procedure is a subroutine for growth processing. It is composed of three parts, the data structure, initialization, and the growth process.

The data structure (see Figure 14) is defined as local and global variables. Global variables are the same as the data structure in Figure 11 with the exception of the INPORT[node] array. INPORT[] is used for keeping track of the INBOUND port status.

```
/*************************************************************/
/*      LOCAL AND GLOBAL VARIABLES DECLARATION               */
/*************************************************************/

        growth()
               {
                /* define external variables */
                extern int NNODES,NPORTS;
                extern int NODE_CNN[][7];
                extern int PORT_CNN[][7];
                extern int PORT_STATUS[][7];
                extern int LINK_STATUS[][7];
                extern int GROWTH_FIFO[33];
                extern int NODE_ROOT[33];
                extern int NET_MEMBER[33];
                extern int INPORT[33];

                /* define variables */
                       int INSERTION_POINT;
                       int EXTRACTION_POINT;
                       int GROWTH_POINT;
                       int NEXT_NODE;
                       int NODE, PORT;
                       int INIT = -9;
                       int i,j;
```

Figure 14 Data Structure Listing Code for Local and Global Variables in Growth Procedure.

Local variables for the growth procedure use internal subroutines for keeping track of pointers and are declared as integer type. The local variables are:

INSERTION_POINT	;	queue tail pointer of the GROWTH_FIFO for the port growth iteration.
EXTRACTION_POINT	;	queue head pointer of the GROWTH_FIFO for the node growth iteration.
GROWTH_POINT	;	pointer to the current growth point of the GROWTH_FIFO.
NEXT_NODE	;	pointer of the destination node.
NODE, PORT	;	pointers of nodes and ports in DSPM network.

(node = Ø -> NNODES, port = Ø -> NPORTS; NNODES = number of nodes - 1, NPORTS = number of ports - 1).

The initialization part of the growth procedure (see Figure 15) is the first step in the growth procedure. In this step pointers to the various tables of the data structure are initialized. The contents of NODE_ROOT[node] and GROWTH_FIFO[node] are set to their initial conditions (INIT). NET_MEMBER[node] is initialized by setting the contents to identify no members (NOT_A_MEMBER). The PORT_STATUS array is set to inactive status on all ports except the bus controller ports which are set to OUTBOUND status if the link status is not USELESS. If a port is marked USELESS, the port status is set to inactive status. The local variable pointers are initialized by setting the INSERTION_POINT, EXTRACTION_POINT, and GROWTH_POINT to "Ø".

```
/*************************************************************/
/*          INITIALIZE  DATA STRUCTURE                       */
/*************************************************************/

        for ( NODE = Ø; NODE <= NNODES; NODE++ )
           {
            NODE_ROOT[NODE] = INIT;
            GROWTH_FIFO[NODE] = INIT;
            NET_MEMBER[NODE] = NOT_A_MEMBER;

            for ( PORT = Ø; PORT <= NPORTS; PORT++ )
                 PORT_STATUS[NODE][PORT] = INACTIVE;
                 /* next port */
           }  /* end for node */

        /* initialize pointer */
        INSERTION_POINT = Ø;
        EXTRACTION_POINT = Ø;

        for ( PORT = Ø; PORT <= NPORTS; PORT++ )
           {
            if ( LINK_STATUS[BUS_CONT][PORT] != USELESS )
              PORT_STATUS[BUS_CONT][PORT] = OUTBOUND;
           }  /* end for port */

        /* initialize GROWTH_POINT = BUS_CONT */
        GROWTH_POINT = BUS_CONT;
        GROWTH_FIFO[INSERTION_POINT] = Ø;

        /* increment insertion point */
        INSERTION_POINT += 1;
```

Figure 15 Source Code Listing for Initialization Part of Growth Procedure

Following the flowchart, the GROWTH_FIFO[INSERTION_POINT] starts at the bus controller and increments the pointer (INSERTION_POINT) by one. The growth process (see Figure 16) part has two "For" loops, an outer loop for the node growth iteration loop and an inner loop for the port iteration loop.

```
/*************************************************************/
/*                      GROWTH PROCESS                       */
/*************************************************************/

for ( NODE = 0; NODE <= NNODES; NODE++ )
   {
    for ( PORT = 0; PORT <= NPORTS; PORT++ )
       {
        if ( LINK_STATUS[GROWTH_POINT][PORT] != USELESS )
          {
           NEXT_NODE = NODE_CNN[GROWTH_POINT][PORT];
           if ( NET_MEMBER[NEXT_NODE] == NOT_A_MEMBER )
             {
              if ( GROWTH_POINT == BUS_CONT )
                 NODE_ROOT[NEXT_NODE] = PORT;
              else
                {
                 NODE_ROOT[NEXT_NODE] =
                 NODE_ROOT[GROWTH_POINT];
                 PORT_STATUS[GROWTH_POINT][PORT] = OUTBOUND;
                } /*end if growth_point = bus_cont */
              if ( CONFigure(GROWTH_POINT,NEXT_NODE) )
                {
                 INPORT[NEXT_NODE] =
                 PORT_CNN[GROWTH_POINT][PORT];
                 NET_MEMBER[NEXT_NODE] = MEMBER;
                 GROWTH_FIFO[INSERTION_POINT] = NEXT_NODE;
                 INSERTION_POINT += 1;
                } /* end conFigure */
             }  /* end if not a member */
          }  /* else if NEQ useless */

        else if ((NODE_CNN[GROWTH_POINT][PORT] == 0) &
                   (PORT_STATUS[BUS_CONT][(PORT_CNN
                    [GROWTH_POINT][PORT])] == OUTBOUND))
                PORT_STATUS[GROWTH_POINT][PORT] = INBOUND;
          /* end if NEQ useless */
       }  /* end for  port */

    GROWTH_POINT = GROWTH_FIFO[EXTRACTION_POINT];
    EXTRACTION_POINT += 1;

   }  /* end for node */

}  /* end growth process */
```

Figure 16 Source Code for the Growth Process

The growth process starts with the bus controller for the node growth iteration and with port Ø for the port growth iteration. The first growth iteration begins by fetching the new growth point and checking the link status to determine whether or not the link is USELESS. If the link is USELESS, the routine branches to check the node connection array and sets the port status to INBOUND if the node is already connected to the bus controller. The algorithm then goes to the next port and starts the growth iteration over again. If a port is usable, the routine checks the network membership table (NET_MEMBER(NEXT_NODE) = MEMBER, or NOT_A_MEMBER) to determine whether the port connects to a destination node (NEXT_NODE) that is already active or to an inactive successor node. If the port connects to an active node, the port can not be activated, the routine branches to the next port and starts next growth iteration. If the successor node is inactive, the connection can be made, and the node root of the successor node must be determined (NODE_ROOT[NEXT_NODE] = port, or NODE_ROOT[GROWTH_POINT]). At the same time the port status for the growth point node port is set to OUTBOUND. Finally, "CONFIGURE[GROWTH_POINT,NEXT_NODE]" the protocol module must be called to check node availability for connection. If the CONFIGURE routine returns a successful result, PORT_STATUS of the successor node's port is set as INBOUND by the CONFIGURE routine. After the INPORT, NET_MEMBER, GROWTH_FIFO and INSERTION_POINT data structures

have been manipulated as in Figure 16, the port pointer is incremented to point to the next port and the next growth iteration starts. If there are no more ports, the current growth iteration is complete. The new GROWTH_POINT is obtained by incrementing the GROWTH_FIFO's EXTRACTION_POINT. After the data structure is manipulated, the growth routine starts its iterative process at the next node. If there are no more nodes available, the growth procedure routine is exited.

E. DEVELOPED TOOL AND SIMULATION

The tools developed for the GROWTH algorithm have led to the previously-described data structures for manipulation and tracing of data during execution. The software tools are written in modules and have been designed to create a test environment as close to a reality as possible. The modules test their own operation, the interaction between modules, as well as the data. In addition, the modules are capable of interactive operation which allows a user to more efficiently test the software. Interactive operation is important since previously unobtainable information contained in the dynamics of the data structures is displayed on the computer monitor as the system operates interactively. The display is periodically updated so the user can quickly grasp what errors have occurred and where they are. Such errors are more easily identified with the input level than in the process that has been executed.

Simulation is performed by combining the previously-assembled and constructed modules to execute the entire process. This illustrates how the data structure of the GROWTH algorithm is manipulated.

The availability of detailed information on the data structure as it is being modified is critical to the debugging process because if the incoming data is invalid, the entire simulation run is ruined. The use of structured, high-level C programming language and its powerful file system support greatly ease the development of individual modules, as well as future modifications to such modules, there by, reducing development time.

Figures 17 and 18 show the source code listing of the simulation program. The simulation program is composed of three main processes, MENU(), CRPL(), and GROWTH().

MENU() is the main interactive program module, used for simulation, it consists of two processor modules and two routines. The two processor modules are INWRTFN() and RDFILE(). Both are input/output modules that use floppy disks to store data structures for the DSPM network. The two routines are CHECKNCN() and CHANGEDATA(). MENU() is designed as an interactive program for input from the keyboard and transfer of output data to and from a floppy disk or harddisk. MENU() also has properties to check the input data for errors and the ability to change any data as necessary (see detail in Appendix B).

CRPL() [Appendix C] is a program module designed to initialize LINK_STATUS and PORT_STATUS before their use in the growth process. CRPL() consists of two routines, PRPCN() and PRNLK() [both are in Appendix J].

```
/*************************************************************/
/*      PROGRAM SIMULATE GROWTH ALGORITHM FOR DSPM           */
/*************************************************************/

                #include <stdio.h>
                #include <fcntl.h>
                #include <io.h>
                #include <types.h>

                #define  INACTIVE      0
                #define  INBOUND       1
                #define  OUTBOUND     -1
                #define  USELESS       2
                #define  FAILED        1
                #define  MEMBER        1
                #define  NOT_CONN     -1
                #define  NOT_A_MEMBER  0
                #define  BUS_CONT      0

                #define  NUMNODES     NNODES+1
                #define  NUMPORTS     NPORTS+1

                int      NODE_CNN     [33][7];
                int      PORT_CNN     [33][7];
                int      LINK_STATUS  [33][7];

                int      GROWTH_FIFO  [33];
                int      NODE_ROOT    [33];
                int      NET_MEMBER   [33];
                int      INPORT       [33];

                int      NNODES,NPORTS;

                char     *pathname    [15];
```

Figure 17 Source Code Program Simulation Part I

```
main()
     {
      extern int  NNODES,NPORTS;
      extern int  NODE_CNN    [][7];
      extern int  PORT_CNN    [][7];
      extern int  PORT_STATUS[][7];
      extern int  LINK_STATUS[][7];
      extern int  GROWTH_FIFO [33];
      extern int  NODE_ROOT   [33];
      extern int  NET_MEMBER  [33];
      extern int  INPORT      [33];
      extern char *pathname   [15];
      int         c,i,j,n,p,z;

/*   DISPLAY SELECTION MENU FOR NODE CONNECTION ARRAY      */
do{
   printf("\n\tPROGRAM SIMULATE GROWTH ALGORITHM\n\n\n");
   printf("\n\tENTER # FOR SELECTION:\n\n");
   printf("\n\t\t--> 1 <-- MENU SELECTION NODE CONNECTION ");
   printf("\n\t\t--> 2 <-- SIMULATE GROWTH ALGORITHM ");
   printf("\n\t\t--> n <-- OR OTHERS KEY TO EXIT\n\n\n\n\n");
   printf("\n\tENTER # SELECTION = ");
   z = getchar();
   c = z;
   while ((c == '\n') || ( c == EOF))
        c = getchar();
   switch (c)
        {
         case '1':
   /* subroutine menu selection */
                menu();
   /* go back to simulate prompt */
                c = 1;
                break;
         case '2':
   /* create port connection and link status routine */
                crpl();
   /* growth algorithm subroutine */
                growth();
   /* go back to simulate prompt  */
                c = 1;
                break;
          default:
                c = 0;
                break;
        } /* end switch */
  } while (c == 1); /* end do while */
} /* end simulation */
```

Figure 18 Source Code Program Simulation Part II

GROWTH() [Appendix D] is the same as the GROWTH algorithm but is designed for interactive operation of checking, testing, and displaying the output of simulation results.

More detailed discussion of each module and routine may be found in Appendices [A] through [J].

III. A PROPOSED METHOD OF PERFORMANCE EVALUATION

Within the constraints of the current growth process, the performance evaluation is based solely on the time required to execute the software routines assuming that the configure routine is supplied with only a success or failure indication for the communication protocol (This does not include time delays in communication).

In this scenario, the performance of the DSPM system, as it grows from its initial state to its fully configured state, is measured without regard to the communication process of the DSPM network. However, from a total performance point of view we are interested not only in the performance of software but also the performance of the communication protocol.

The proposed concept of performance evaluation for the GROWTH algorithm is provided in two parts:

1. The first part is the GROWTH algorithm which considers data structure manipulation without considering the performance of communication. The configure module considered is given a communication output and only determines whether there success or failure.

2. The second part is the performance of the communication protocol.

A. DATA STRUCTURE PERFORMANCE EVALUATION

The performance of the GROWTH algorithm software can be evaluated by computation and measurement of each

instruction's execution time. The time performance can be computed in terms of growth time under various fault conditions. A modification to the GROWTH algorithm can be quantified by the same spectrum of possible fault conditions. The general concept of performance evaluation can be illustrated by using the electrical terminology and Kirchoff's current law. By representing the control flow of a program as current, one can draw a signal flow graph where the main flow of the software module is the circuit current loops.

For example, we have a program which is the summation of input A(i) where i is integer Ø to 1Ø23 (1Ø24 iterations). A source program of an example is shown below:

```
int  A[1Ø24]  /* define input array 1Ø24 elements  */
int  SUM = Ø; /* INITIALIZE SUM = Ø  */
main()
    {
     For ( i = Ø; i < 1Ø24 ; i++ )
         SUM = SUM + A[i];
    } /* end for loop  */
```

Figure 19 shows the flow chart and control flow graph of an example program. Letting t_i be the execution time associated with each arc, one can see that the total execution time equals the sum of the products of $t_i * y_i$, where Y_i is equal to the number of arcs executed. We can write the equation of total execution time as:

$$E_t = \sum_{i=0}^{n} t_i y_i \qquad \text{.....(1)}$$

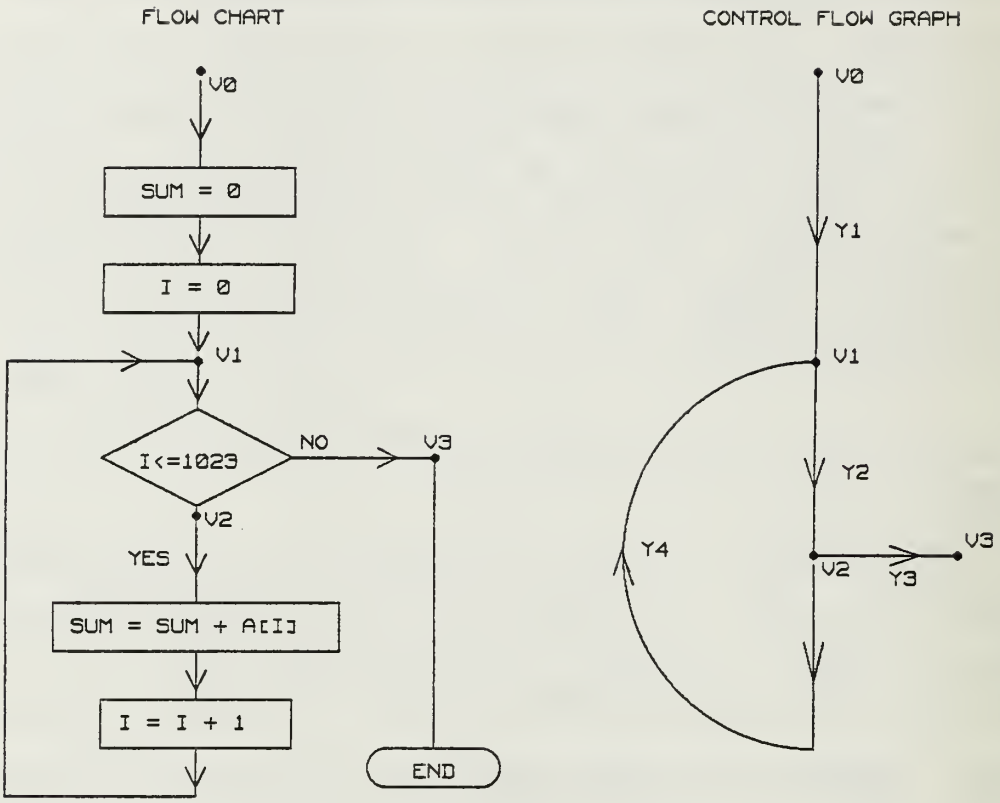


Figure 19 An Example Flow Chart and Control Flow Graph

From the control flow graph we can construct a spanning tree structure of the connected arcs. From this representation we can write the dependent and independent arcs in the matrix form (see Figure 20). By writing dependent arcs in terms of independent arcs, the number "1" represents the arc whose directed edge is in the same direction of flow, and "-1" represents the arc whose directed edge is in the opposite direction, and "0" is for arcs which

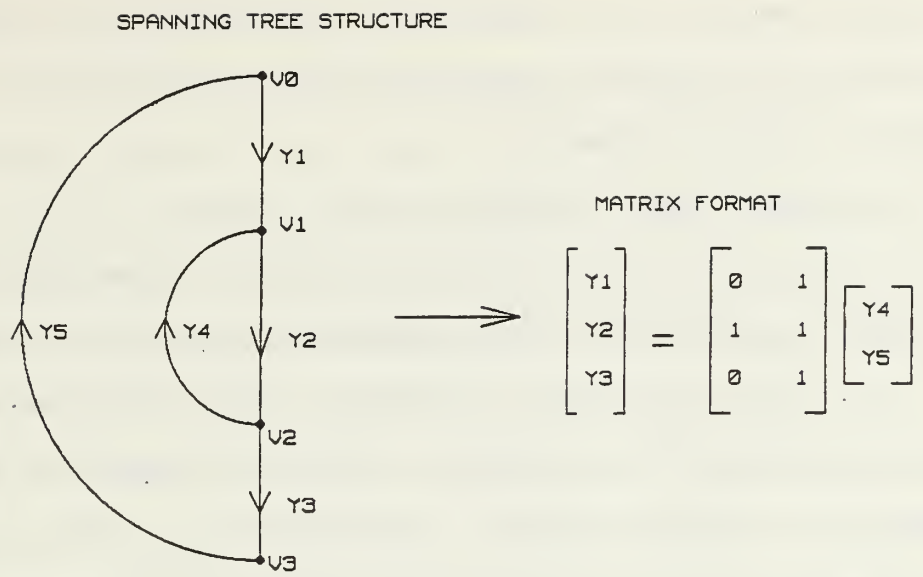


Figure 20 An Example of Spanning Tree Structure and Matrix Representation

are not in the loop. Using the equivalent Kirchoff's current law we can substitute into equation 1 and show the following:

$$E_t = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 & t_5 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \end{bmatrix}$$

$$= \begin{bmatrix} t_1 & t_2 & t_3 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} + \begin{bmatrix} t_4 & t_5 \end{bmatrix} \begin{bmatrix} Y_4 \\ Y_5 \end{bmatrix} \quad \text{.....(2)}$$

From this example, it can be seen that y_4 and y_5 are independent arcs and y_1, y_2 and y_3 are dependent arcs. Y_5 is similar to a current source in a circuit loop and is shown as a dashed line in Figure 20. The variable y_4 equals 1024 which is the number of iterations executed in the inner loop as shown in the spanning tree structure. The variable y_5

obviously equals 1 iteration for the end of the program and/or a return back to the beginning. The solution of this example is of the form:

$$E_t = (t_2+t_4)1024 + (t_1+t_2+t_3+t_5)1 \quad \ldots\ldots\ldots\ldots\ldots\ldots(3)$$

and we can evaluate the total execution time by measurement the execution time for each instruction for each value of t_i in the above equation. However, the latency times due to delay in communication network are not modeled in the above equation. Incorporating communication delays into the performance evaluation model required a detailed model of the communication network and its protocol. In the case of the GROWTH algorithm that detailed model would be used to provide not only success/failure status to the configure routine but latency times for the communication process. The Time Petri net is one such model.

B. TIMED PETRI NET PROTOCOL MODEL

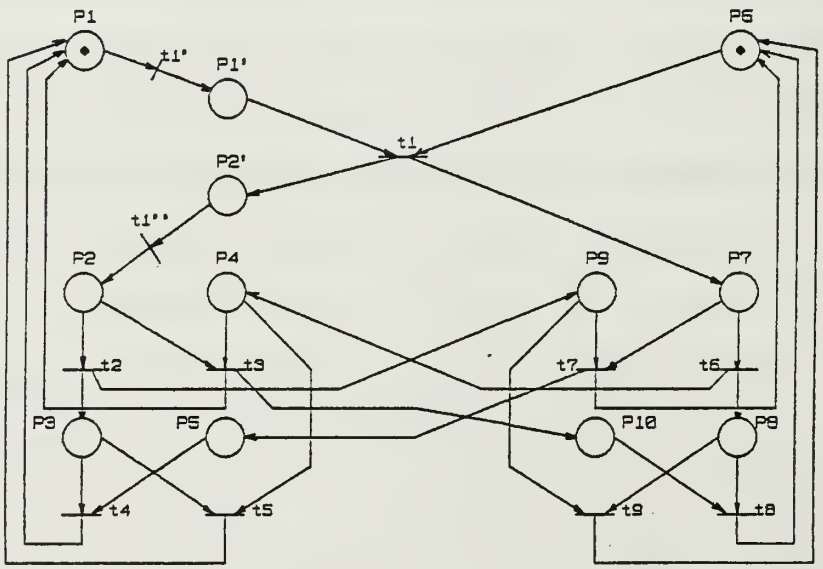
During the past decade, the notion of time has been added to Petri net models. The Timed Petri net can now use the time parameter to effectively describe and quantitatively specify the performance of a communication protocol. The addition of time as a parameter also makes it easier to do performance evaluation and analysis by computers because Timed Petri nets are capable of describing concurrency conflicts and synchronizations of processes. Now Petri nets have become very popular for representing Distributed Computer Systems.

Timed Petri Nets (TPN) have been used for illustration by K. Garg [Ref. 9] at the European Computer Manufacture's Association (ECMA) for the transport protocol in the International Standard Organization (ISO) [Ref. 10]. For fixed time models (fixed delay times) and a given set of relations describing the behavior of such a TPN, the model's maximum computation rate can be found [Ref. 11].

The European Computer Manufacturer Association (ECMA) transport protocol is an end to end protocol standing between the session and the network layers of the ISO reference model [Ref. 10]. The transport layer provides the means to communicate by establishing links between two transport entities. Processes can be programmed to directly access transport services without going through session and presentation layers of the ISO reference model.

The TPN concept used in point-to-point communications is illustrated by K. Garg to be in connection and disconnection phases. There are no mechanisms for recovery from errors. When an error occurs, each end returns to an initial disconnected state and all messages are ignored but the requests for communication. Transport entities are managed locally in such a way that a transport entity can only be involved in a transport connection if it is disconnected. That is, it has not tried to establish or has not already established a transport connection. Two transport entities communicate by first establishing a

transport connection. Once a connection is established, between initially unconnected entities, one is specified as the initializer and the other as the acceptor. Hence, simultaneous attempts at connection result in the creation of two transport connections, one an initiating entity and the other an accepting entity [Ref. 12]. The TPN model of entity behavior in connection and disconnection phase is shown in Figure 21 [Ref. 11].



PLACES

State of one entity:

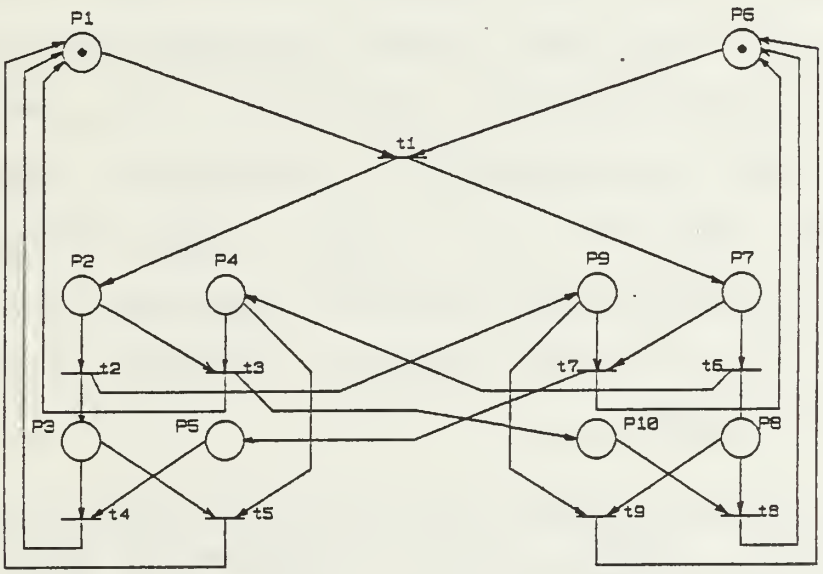
P1 and P6 : idle
p1' : waiting connect confirm
P2' : connection confirm
P2 and P7 : data transfer
P3 and P8 : waiting disconnect confirm
P4 and P9 : disconnection request
P5 and P10 : disconnection confirm

TRANSITIONS

t1' : connection request
t1'' : connection established
t1 : connection accepted
t2 and t6 : request for disconnection
t3 and t7 : disconnection accepted
t4 and t8 : disconnect end
t5 and t9 : disconnection accepted

Figure 21 TPN Connection-Disconnection Phase [Ref. 11]

We can use the same approach to design simulation protocol models to evaluate performance specification of communication protocol by using Timed Petri net model as K. Garg represented in Reference 9. The TPN model for performance evaluation as illustrated by K. Garg is shown in Figure 22 [Ref. 9]. The configure protocol model can be designed individually and is used to evaluate the time performance of the DSPM network communication in conjunction with the GROWTH algorithm.



PLACES

State of one entity:

P1 and P6 : idle
P2 and P7 : data transfer
P3 and P8 : waiting disconnect confirm
P4 and P9 : disconnection request
P5 and P10 : disconnection confirm

TRANSITIONS

t1 : connection accepted
t2 and t6 : request for disconnection
t3 and t7 : disconnection accepted
t4 and t8 : disconnect end
t5 and t9 : disconnection accepted

Figure 22 The TPN Model for Performance Evaluation [Ref. 11]

IV. CONCLUDING REMARKS

Incorporating performance evaluation for the DSPM communication process into a software performance evaluation tool would allow total system performance to be established prior to building an expensive prototype system.

The goal of this thesis is to develop a GROWTH algorithm for the Dispersed Sensor Processing Mesh (DSPM) network which will manage a fault-tolerant computer communication network. Simulation tools are developed which simulate data structure in order to verify and validate the operation of the GROWTH algorithm. The performance concepts of the communication network were investigated to develop an approach for performance evaluation on a DSPM type network. The communication performance evaluation approach is based on TPN concept. The software simulation program is composed of three main processes:

1. Menu driven program having many features for I/O data transfer between storage devices as well as the ability to select options for modification and/or correction of the data input information to the DSPM network.
2. Create and initialize data structure program from data input information of the DSPM network.
3. Growth process program.

The software code is written in standard high level C-language to maintain a portable software module for use on

different computer system and/or different operation systems with a minimum number of modifications. Despite technology changes in physical hardware which will occur in the future, the software modules will remain valid due to the inherent flexibility of the C-language. Software modules can be tested to allow an assessment as to the viability of the GROWTH algorithm and general DSPM approach even before the hardware exists.

Practical implementation of software performance evaluations for the GROWTH algorithm can be used as a system development tool. One example of such a system is the HP64000, logic development system emulator, which provides processor run controls and simulation environment to download and execute software at operating speeds for analysis and debugging. Software modules can be tested even before hardware of DSPM node is constructed. Typically subroutines and modules of code to be investigated by the HP64000 are entered as measurement events by name or symbol. Addresses and range can be used to set up performance measurements and which allows the HP64000 to look directly at interactions between subroutines. We can control and compare execution times of key modules, and can determine best-case and worst-case times as well as where time limitations impede smooth program flow. A system development such as this will show us what kind of performance to expect. Available HP64000 software development tools can be used directly with

subroutine modules written in the C-language and with the addition of C compiler, modules can be linked to form a single program. Microprocessors which are supported by the emulator processor can be selected from the popular microprocessor families. More generally, this conceptual approach could be applied to other communication bus base architectures.

APPENDIX A

HOW TO USE SIMULATION PROGRAM

The simulation program is grouped into easily managed operations which are available in the execution file called "simulation.exe". Computer assumes the use of an MS-DOS compatible microcomputer. The steps required to run the simulation program are described as follows:

1. Load MS-DOS operation system from a system disc.
2. Run the simulation program "simulation.exe" which is provided on either floppy disk or harddisk devices as below.

```
c> simulation <cr>
```

The display screen prompts the main menu:

```
PROGRAM SIMULATE GROWTH ALGORITHM
ENTER # FOR SELECTION:

        --> 1 <-- MENU SELECTION NODE CONNECTION
        --> 2 <-- SIMULATE GROWTH ALGORITHM
        --> N <-- OR OTHERS KEY TO EXIT

ENTER # SELECTION =
```

3. Selection option 1 (go to submenu for input node connection array and DSPM specification. See more details in Appendix B).

ENTER # SELECTION = 1<cr>

SELECT INPUT DATA DSPM STRUCTURE
ENTER # FOR SELECTION:

--> 1 <-- CREATE NEW DATA AND WRITE DATA FILE
--> 2 <-- READ DATA FILE AND OR MODIFY FILE
--> 3 <-- READ DATA FILE AND CONTINUE
--> n <-- OR OTHERS KEY TO CONTINUE

---

4. Selection option 2 (main menu), assuming we already have the node connection array from the submenu above and go back to main menu.

For example, using node connection array for the DSPM6 shown in Figure 12a. The display screen will show the port connection array and the link status array which are outputs of the crpl() module (see more details in Appendix C), and then the program branches to the growth() module (see more details in Appendix D). For example using the node connection array from figure 12a simulates the case of the fault free network in which the configure program returns successful node communications. The results of the simulation in this example are shown the output data structure of the growth algorithm. The display screen will show array variables and the port status array for showing

the correctness of DSPM data structure. The program will then go back to main menu.

ENTER # SELECTION = 2<cr>

PORT CONNECTION

NODE	PORT 0	1	2	3
0	0	0	0	-1
1	0	3	0	0
2	3	3	1	-1
3	2	-1	2	1
4	1	3	0	1
5	2	2	2	-1
6	2	2	1	-1

HIT enter KEY TO CONTINUE
<cr>

LINK STATUS

NODE	PORT 0	1	2	3
0	0	0	0	2
1	2	0	0	0
2	0	0	0	2
3	0	2	0	0
4	2	0	0	0
5	0	0	0	2
6	2	0	0	0

HIT enter KEY TO CONTINUE
<cr>

VARIABLES ARRAY

NODE	NET_MEMBER	INPORT	GROWTH_FIFO
Ø	Ø	Ø	Ø

HIT enter KEY TO CONTINUE
<cr>

VARIABLES ARRAY

NODE	NET_MEMBER	INPORT	GROWTH_FIFO
Ø	Ø	Ø	Ø
1	1	Ø	1

HIT enter KEY TO CONTINUE
<cr>

VARIABLES ARRAY

NODE	NET_MEMBER	INPORT	GROWTH_FIFO
Ø	Ø	Ø	Ø
1	1	Ø	1
2	1	Ø	4

HIT enter KEY TO CONTINUE
<cr>

(and so on until node 6)

VARIABLES ARRAY

NODE	NET_MEMBER	INPORT	GROWTH_FIFO
Ø	Ø	Ø	Ø
1	1	Ø	1
2	1	Ø	4
3	1	Ø	6
4	1	Ø	3
5	1	Ø	2
6	1	Ø	5

HIT enter KEY TO CONTINUE
<cr>

```
                    PORT STATUS
                    PORT
                    Ø       1       2       3
NODE
  Ø               -1      -1      -1       Ø
  1                1       Ø      -1      -1
  2                Ø       Ø       Ø       Ø
  3                Ø       Ø       Ø       Ø
  4                1       Ø      -1       Ø
  5                Ø       Ø       Ø       Ø
  6                1       Ø       Ø       Ø


        HIT enter KEY TO CONTINUE
        <cr>


PROGRAM SIMULATE GROWTH ALGORITHM
ENTER # FOR SELECTION:

          --> 1 <-- MENU SELECTION NODE CONNECTION
          --> 2 <-- SIMULATE GROWTH ALGORITHM
          --> N <-- OR OTHERS KEY TO EXIT

ENTER # SELECTION =
```

---

5. Selection option 3 (main menu) is obviously for exit from the program.

APPENDIX B

INPUT DATA STRUCTURE MODULE

The MENU() module is designed to operate interactively with the input from keyboard and the output to storage devices. The display screen shows menu options as below.

```
SELECT INPUT DATA DSPM STRUCTURE
ENTER # FOR SELECTION:

        --> 1 <-- CREATE NEW DATA AND WRITE DATA FILE
        --> 2 <-- READ DATA FILE AND OR MODIFY FILE
        --> 3 <-- READ DATA FILE AND CONTINUE
        --> n <-- OR OTHERS KEY TO CONTINUE

ENTER # SELECTION =
```

The operations of this module are as following:

1. Input the DSPM specifications and node connection array from the keyboard for the global variable array in the simulation program and output the data information to the storage devices (floppy disk and/or harddisk) by creating a data file as specified from the keyboard input. This is done by selecting option 1 in menu. This operation is executed by submodule inwrtfn() (see more details in Appendix E).

2. Read and/or modify and write the data file back, or create the new data file from the DSPM input specifications and the node connection array from the specified storage devices. These operations are in option selections 2 and 3 of submodules inwrtfn() and rdfile() [Appendix F], and routine changedata() [Appendix H].
3. Specification of the DSPM network is designed to have a minimum of 2 nodes, a maximum of 33 nodes, a minimum of 2 ports, and a maximum of 7 ports for each node.
4. Error detection is designed to discover errors during input from the keyboard and/or to check data information from storage devices. Data can be modified or corrected in an interactive manner. These operations are executed by routines changedata() [Appendix H] and checkncn() [Appendix I].
5. Specification data error detection where the input data is checked against the system design specification, and self check node to validate potential physical links.
6. Any input in the input data file array can be modified on an individual element basis or the whole data array can be redone from scratch.

These modules can be run individually and/or linked together by the compiler to be a simulation program. Each module can be compiled to object code and linked together or

combined into the same program, by replacing the subroutine name with main() and compiling and running separately.

Example 1. Select option 1 from the main menu and select option 1 of the submenu (menu selection node connection) and enter:

- "a:dspm6.dat"

  a: --> device storage A

  dspm6.--> filename

  dat --> file type

- "5" (for 6 DSPM nodes)
- "4" (for 5 ports per DSPM node)

```
PROGRAM SIMULATE GROWTH ALGORITHM
ENTER # FOR SELECTION:

  --> 1 <-- MENU SELECTION NODE CONNECTION
  --> 2 <-- SIMULATE GROWTH ALGORITHM
  --> n <-- OR OTHERS KEY TO EXIT


ENTER # SELECTION = 1<cr>



SELECT INPUT DATA DSPM STRUCTURE
ENTER # FOR SELECTION:

  --> 1 <-- CREATE NEW DATA AND WRITE DATA FILE
  --> 2 <-- READ DATA FILE  AND OR MODIFY  FILE
  --> 3 <-- READ DATA FILE  AND CONTINUE
  --> n <-- OR OTHERS KEY TO CONTINUE
```

```
ENTER # SELECTION = 1<cr>

PROG. INPUT DATA DSPM STRUCTURE AND WRITE DSPM#.DAT FILE

ENTER NAME OF DATA FILE "[_:_fn_._ft_]" (c:dspm6.dat)
ENTER # OF NODES  Ø TO NNODES  FOR 1 <= NNODES <= 32
ENTER # OF PORTS  Ø TO NPORTS  FOR 1 <= NPORTS <= 6

ENTER NAME   = a:dspm6.dat<cr>

ENTER NNODES = 5<cr>

ENTER NPORTS = 4<cr>


DATA FILE = a:dspm6.dat
NNODES = 5
NPORTS = 4


ANY CHANGE ? [y]   HIT [enter] KEY TO CONTINUE
```

Example2. Change and test input data as follows:

- "y" (want to change data input)
- "c:dspm.dat" (change device A to device C)
- "Ø" (test error input),
- "6" (and change to 7 DSPM nodes)
- "Ø" (test error input),
- "3" (and change to 4 ports per DSPM node)

```
DATA FILE = a:dspm.dat
NNODES = 5
NPORTS = 4
TO BE CHANGED
```

```
ENTER NAME = c:dspm6.dat<cr>

ENTER NNODES = Ø<cr>
ENTER NNODES 1 -> TO 32
----> ERROR <---- try again

ENTER NNODES = 6<cr>

ENTER NPORTS = Ø<cr>
ENTER NPORTS 1 -> TO 6
----> ERROR <---- try again

ENTER NPORTS = 3<cr>

DATA FILE = c:dspm6.dat
NNODES = 6
NPORTS = 3

ANY CHANGE ? [y]   HIT [enter] KEY TO CONTINUE
```

Example 3. No more change and start input node array using the same data structure for the DSPM6 node connection in figure 12a :

- node Ø port Ø connects to node 1
- node Ø port 1 connects to node 4
- node Ø port 2 connects to node 6
- node Ø port 3 is not connected (-1)
- node 1 port Ø connects to node Ø
- node 1 port 1 connects to node 4
- node 1 port 2 connects to node 3
- node 1 port 3 connects to node 2

(and so on)

- node 6 port 0 connects to node 0
- node 6 port 1 connects to node 2
- node 6 port 2 connects to node 5
- node 6 port 3 connects to node 4

```
<cr>

DATA FILE = c:dspm6.dat
NNODES = 6
NPORTS = 3

ENTER NODE CONNECTION ARRAY[7][4]

ENTER NODE_CNN[0,0] = 1<cr>
ENTER NODE_CNN[0,1] = 4<cr>
ENTER NODE_CNN[0,2] = 6<cr>
ENTER NODE_CNN[0,3] = -1<cr>

                    NODE CONNECTION
                    PORT
                    0       1       2       3
NODE
  0                 1       4       6      -1

ANY CHANGE IN NODE[0->0] PORT[0->3] ? [y]
HIT [enter] KEY TO CONTINUE

                    NODE CONNECTION
                    PORT
                    0       1       2       3
NODE
  0                 1       4       6      -1
```

```
ENTER NODE_CNN[1,Ø] = Ø<cr>
ENTER NODE_CNN[1,1] = 4<cr>
ENTER NODE_CNN[1,2] = 3<cr>
ENTER NODE_CNN[1,3] = 2<cr>
```

	NODE CONNECTION PORT			
	Ø	1	2	3
NODE				
Ø	1	4	6	-1
1	Ø	4	3	2

```
ANY CHANGE IN NODE[Ø->1] PORT[Ø->3] ? [y]
HIT [enter] KEY TO CONTINUE
```

	NODE CONNECTION PORT			
	Ø	1	2	3
NODE				
Ø	1	4	6	-1
1	Ø	4	3	2

..... and so on .....

	NODE CONNECTION PORT			
	Ø	1	2	3
NODE				
Ø	1	4	6	-1
1	Ø	4	3	2
2	1	3	6	-1
3	1	-1	5	2
4	Ø	6	5	1
5	4	6	3	-1

```
ENTER NODE_CNN[6,Ø] = Ø<cr>
ENTER NODE_CNN[6,1] = 2<cr>
ENTER NODE_CNN[6,2] = 5<cr>
ENTER NODE_CNN[6,3] = 4<cr>
```

```
                    NODE CONNECTION
                    PORT
                    Ø       1       2       3
NODE
  Ø                 1       4       6      -1
  1                 Ø       4       3       2
  2                 1       3       6      -1
  3                 1      -1       5       2
  4                 Ø       6       5       1
  5                 4       6       3      -1
  6                 Ø       2       5       4

DO YOU WANT ANY CHANGE?

*--->ENTER  [a] TO CHANGE ALL AT THE BEGINNING
            [b] TO CHANGE AT EACH NODE_CNN[][]

HIT [enter] KEY OR ANY CHARACTER TO CONTINUE
```

Example 4. Assume the input node connection array is done and there is no error detected, and no more changes. The program will write a data file to the storage device as specified (device:filename.filetype) and then go back to the main menu.

```
                    NODE CONNECTION
                    PORT
                    Ø       1       2       3
NODE
  Ø                 1       4       6      -1
  1                 Ø       4       3       2
  2                 1       3       6      -1
  3                 1      -1       5       2
  4                 Ø       6       5       1
  5                 4       6       3      -1
  6                 Ø       2       5       4
```

```
DATA FILE   =  c:dspm6.dat
NNODES      =  6
NPORTS      =  3

HIT [enter] KEY TO CONTINUE
```

Example 5. Error detection during input of the node connection.

```
                    NODE CONNECTION
                    PORT
                    Ø        1        2        3
NODE
  Ø                 1        4        6       -1


ENTER NODE_CNN[1,Ø] = -2<cr>

ENTER NODE_CNN -1 -> TO 6
----> ERROR <---- try again



ENTER NODE_CNN[1,Ø] = 7<cr>

ENTER NODE_CNN -1 -> TO 6
----> ERROR <---- try again


ENTER NODE_CNN[1,Ø] = Ø<cr>
ENTER NODE_CNN[1,1] = 4<cr>
ENTER NODE_CNN[1,2] = 3<cr>
ENTER NODE_CNN[1,3] = 2<cr>
```

```
                    NODE CONNECTION
                    PORT
                    Ø       1       2       3
NODE
  Ø                 1       4       6      -1
  1                 Ø       4       3       2

ANY CHANGE IN NODE[Ø->1] PORT[Ø->3] ? [y]
HIT [enter] KEY TO CONTINUE
```

Example 6. Assume there are 3 errors in node connection after input finished:

- node 1 port 3 = -1 (should be 2)
- node 5 port 3 = 1 (should be -1)
- node 6 port 1 = 2 (should be Ø)

The display screen will show possible node connection errors.

```
--->NODE[6] ERROR CONNECTION<----------*
....NODE_CNN[Ø][2]-*/*->NODE[6].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..

--->NODE[1] ERROR CONNECTION<----------*
....NODE_CNN[2][Ø]-*/*->NODE[1].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..

--->NODE[6] ERROR CONNECTION<----------*
....NODE_CNN[2][2]-*/*->NODE[6].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..

--->NODE[1] ERROR CONNECTION<----------*
....NODE_CNN[5][3]-*/*->NODE[1].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..

NODE[1] ERROR  (EXCESS)  CONNECTION
NODE[1] COUNT = 5  --> > #PORTS
```

```
                    NODE CONNECTION
                    PORT
                    Ø        1        2        3
NODE
  Ø                 1        4        6       -1
  1                 Ø        4        3       -1
  2                 1        3        6       -1
  3                 1       -1        5        2
  4                 Ø        6        5        1
  5                 4        6        3        1
  6                 2        2        5        4


DATA FILE   =  c:dspm6.dat
NNODES      =  6
NPORTS      =  3


ANY CHANGE IN NODE[Ø->6] PORT[Ø->3] ? [y]
ENTER *--> n OR ANY CHARACTER TO CONTINUE
y<cr>


                    NODE CONNECTION
                    PORT
                    Ø        1        2        3
NODE
  Ø                 1        4        6       -1
  1                 Ø        4        3       -1
  2                 1        3        6       -1
  3                 1       -1        5        2
  4                 Ø        6        5        1
  5                 4        6        3        1
  6                 2        2        5        4


DATA FILE   =  c:dspm6.dat
NNODES      =  6
NPORTS      =  3


CHANGING IN NODE[Ø->6] PORT[Ø->3]
ENTER NODE,PORT,NODE_CNN

ENTER NODE = 1<cr>
ENTER PORT = 3<cr>

ENTER NODE_CNN[1,3] = 2<cr>
```

```
--->NODE[6] ERROR CONNECTION<----------*

....NODE_CNN[Ø][2]-*/*->NODE[6].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..


--->NODE[6] ERROR CONNECTION<----------*

....NODE_CNN[2][2]-*/*->NODE[6].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..


--->NODE[1] ERROR CONNECTION<----------*

....NODE_CNN[5][3]-*/*->NODE[1].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..


NODE[1] ERROR  (EXCESS)  CONNECTION
NODE[1] COUNT = 5  --> > #PORTS
```

	NODE CONNECTION PORT			
	Ø	1	2	3
NODE				
Ø	1	4	6	-1
1	Ø	4	3	2
2	1	3	6	-1
3	1	-1	5	2
4	Ø	6	5	1
5	4	6	3	1
6	2	2	5	4

```
DATA FILE   =  c:dspm6.dat
NNODES      =  6
NPORTS      =  3


ANY CHANGE IN NODE[Ø->6] PORT[Ø->3] ? [y]

ENTER *--> n OR ANY CHARACTER TO CONTINUE
y<cr>
```

```
                    NODE CONNECTION
                    PORT
                    Ø        1        2        3
NODE
  Ø                 1        4        6       -1
  1                 Ø        4        3        2
  2                 1        3        6       -1
  3                 1       -1        5        2
  4                 Ø        6        5        1
  5                 4        6        3        1
  6                 2        2        5        4


DATA FILE   =  c:dspm6.dat
NNODES      =  6
NPORTS      =  3

CHANGING IN NODE[Ø->6] PORT[Ø->3]
ENTER NODE,PORT,NODE_CNN

ENTER NODE = 5<cr>
ENTER PORT = 3<cr>

ENTER NODE_CNN[1,3] = -1<cr>


--->NODE[6] ERROR CONNECTION<----------*

....NODE_CNN[Ø][2]-*/*->NODE[6].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..


--->NODE[6] ERROR CONNECTION<----------*

....NODE_CNN[2][2]-*/*->NODE[6].........
....PLEASE RECHECK NODE_CONN[][] AGAIN..



                    NODE CONNECTION
                    PORT
                    Ø        1        2        3
NODE
  Ø                 1        4        6       -1
  1                 Ø        4        3       -1
  2                 1        3        6       -1
  3                 1       -1        5        2
  4                 Ø        6        5        1
  5                 4        6        3       -1
  6                 2        2        5        4
```

```
DATA FILE   =  c:dspm6.dat
NNODES      =  6
NPORTS      =  3

ANY CHANGE IN NODE[Ø->6] PORT[Ø->3] ? [y]
ENTER *--> n OR ANY CHARACTER TO CONTINUE
y<cr>
```

NODE	NODE CONNECTION PORT Ø	1	2	3
Ø	1	4	6	-1
1	Ø	4	3	-1
2	1	3	6	-1
3	1	-1	5	2
4	Ø	6	5	1
5	4	6	3	-1
6	2	2	5	4

```
DATA FILE   =  c:dspm6.dat
NNODES      =  6
NPORTS      =  3

CHANGING IN NODE[Ø->6] PORT[Ø->3]
ENTER NODE,PORT,NODE_CNN

ENTER NODE = 6<cr>
ENTER PORT = Ø<cr>

ENTER NODE_CNN[1,3] = Ø<cr>
```

NODE	NODE CONNECTION PORT Ø	1	2	3
Ø	1	4	6	-1
1	Ø	4	3	-1
2	1	3	6	-1
3	1	-1	5	2
4	Ø	6	5	1
5	4	6	3	1
6	Ø	2	5	4

```
DATA FILE   =  c:dspm6.dat
NNODES      =  6
NPORTS      =  3
```

```
ANY CHANGE IN NODE[Ø->6] PORT[Ø->3] ? [y]
ENTER *--> n OR ANY CHARACTER TO CONTINUE
```

Example 7. Select option 2 from the menu (READ DATA FILE AND OR MODIFY FILE). Option 3 is the same as option 2 but without a modify.

- "2" (READ DATA FILE AND/OR MODIFY FILE)
- "a:dspm6.doc" (test error)

  "a:dspm.dat" (want to change file name)

  "a:dspm6.dat" (correct file name)
- "n" no modify (to modify the same as example 6)
- "y" (want to write the new data file)
- "b:dspm6.dat" (write to device b:)

```
SELECT INPUT DATA DSPM STRUCTURE
ENTER # FOR SELECTION:

  --> 1 <-- CREATE NEW DATA AND WRITE DATA FILE
  --> 2 <-- READ DATA FILE  AND OR MODIFY  FILE
  --> 3 <-- READ DATA FILE  AND CONTINUE
  --> n <-- OR OTHERS KEY TO CONTINUE


ENTER # SELECTION = 2<cr>
```

```
PROGRAM READ DATA DSPM STRUCTURE FILE

ENTER NAME OF DATA FILE "[_:_fn_._ft_]
----> SPECIFY DISK DRIVE A:, B: OR C:
----> AND FILENAME.DAT (C:DSPM6.DAT)


ENTER FILE NAME = a:dspm6.doc<cr>


DATA FILE = a:dspm6.doc

DO YOU WANT TO CHANGE FILE NAME?  [y]
HIT [enter] KEY TO CONTINUE
<cr>


DATA FILE = a:dspm6.doc

NO FILE a:dspm6.doc IN DISK SPECIFIED
************> TRY AGAIN <***********

PROGRAM READ DATA DSPM STRUCTURE FILE

ENTER NAME OF DATA FILE "[_:_fn_._ft_]
----> SPECIFY DISK DRIVE A:, B: OR C:
----> AND FILENAME.DAT (C:DSPM6.DAT)


ENTER FILE NAME = a:dspm.dat<cr>


DATA FILE = a:dspm.dat

DO YOU WANT TO CHANGE FILE NAME?  [y]
HIT [enter] KEY TO CONTINUE
y<cr>


DATA FILE = a:dspm.dat
WANT TO BE CHANGED

ENTER FILE NAME = a:dspm6.dat<cr>


DATA FILE = a:dspm6.dat

DO YOU WANT TO CHANGE FILE NAME?  [y]
HIT [enter] KEY TO CONTINUE
<cr>
```

```
DATA FILE = a:dspm6.dat


                    NODE CONNECTION
                    PORT
                    Ø        1        2        3
NODE
  Ø                 1        4        6       -1
  1                 Ø        4        3       -1
  2                 1        3        6       -1
  3                 1       -1        5        2
  4                 Ø        6        5        1
  5                 4        6        3        1
  6                 Ø        2        5        4


DATA FILE   =   a:dspm6.dat
NNODES      =   6
NPORTS      =   3

HIT [enter] KEY TO CONTINUE

ANY CHANGE IN NODE[Ø->6] PORT[Ø->3] ?  [y]
ENTER *--> n OR ANY CHARACTER TO CONTINUE
n<cr>


DATA FILE = a:dspm6.dat

NNODES = 6
NPORTS = 3


DO YOU WANT TO WRITE THE NEW DATA FILE ?  [y]
HIT [enter] KEY TO CONTINUE
y<cr>


DATA FILE = a:dspm6.dat

NNODES = 6
NPORTS = 3

PROG.  WRITE DATA DSPM STRUCTURE FILE

ENTER NAME OF DATA FILE "[_:_fn_._ft_]
----> SPECIFY DISK DRIVE A:, B: OR C:
----> AND FILENAME.DAT (C:DSPM6.DAT)
```

```
ENTER FILE NAME = b:dspm6.dat<cr>


DATA FILE = b:dspm6.dat

DO YOU WANT TO CHANGE FILE NAME?  [y]
HIT [enter] KEY TO CONTINUE
<cr>


DATA FILE = b:dspm6.dat

NNODES = 6
NPORTS = 3


SELECT INPUT DATA DSPM STRUCTURE

ENTER # FOR SELECTION:

  --> 1 <-- CREATE NEW DATA AND WRITE DATA FILE
  --> 2 <-- READ DATA FILE  AND OR MODIFY  FILE
  --> 3 <-- READ DATA FILE  AND CONTINUE
  --> n <-- OR OTHERS KEY TO CONTINUE
```

```
/*************************************************************/
/*                                                           */
/*    PROGRAM   SUBROUTINE MENU FOR DSPM DATA STRUCTURE      */
/*    FILE NAME SUB_MENU.C  [MENU()]                         */
/*                                                           */
/*************************************************************/

/* PROGRAM REQUIRES 2 SUBROUTINES AND 3 ROUTINES:
   INWRTFN()      (input and create file dspm#.dat subroutine)
   RDFILE()       (read data file dspm#.dat subroutine)
   CHECKNCN()     (node connection array error checking
                   routine)
   CHANGEDATA()   (node connection array changing routine)
   WRTFILE()      (write data file routine) */

#include <stdio.h>

#define  NUMNODES  NNODES+1
#define  NUMPORTS  NPORTS+1
/* define connection array 2 dimensions */
   int   NODE_CNN [33][7];
```

```
/* define variables */
   int   NNODES,NPORTS;
   char  *pathname[15];

menu()
    {
     extern int  NODE_CNN[][7];
     extern int  NNODES,NPORTS;
     extern char *pathname[15];

     /* define variables */
     int c,z;

/* display selection MENU for node connection array */
printf("\n\n\n\n\n\n\n\n\n\n");
printf("\n\n\n\n\n\n\n\n\n\n");
printf("\n\tSELECT INPUT DATA DSPM STRUCTURE \n\n\n");
printf("\n\tENTER # FOR SELECTION:");
printf("\n\t\t--> 1 <-- CREATE NEW DATA AND WRITE DATA
                          FILE");
printf("\n\t\t--> 2 <-- READ DATA FILE AND OR MODIFY FILE ");
printf("\n\t\t--> 3 <-- READ DATA FILE AND TO BE CONTINUE");
printf("\n\t\t--> n <-- OR OTHERS KEY TO BE CONTINUE\n ");
printf("\n\n\n\n\n\n\n\n\n\n");
printf("\n\tENTER # SELECTION = ");

z = getchar();
c = z;
while ((c == '\n') || ( c == EOF))
     c = getchar();
switch (c)
      {
       case '1':

/* subroutine input data NODE_CNN[][] and create dspm#.dat */
             inwrtfn();

/* go back to menu program */
             menu();
             break;
       case '2':

/* subroutine read data node_cnn[][] from dspm#.dat file */
             rdfile();

             /* error checking routine */
             checkncn();

printf("\n\nANY CHANGE IN NODE[0->%d] PORT[0-> %d] ?
        [y]\n",NNODES,NPORTS);
```

```
printf("\nENTER *--> n OR ANY CHARACTER TO BE CONTINUE\n");

z = getchar();
c = z;
while ((c == '\n') || ( c == EOF))
      c = getchar();
      switch (c)
            {
             case 'y':

             /* changing data structure routine */
                changedata();

             default:
                printf("\n\n\n\n\n\n\n\n\n\n");
                printf("\n\n\n\n\n\n\n\n\n\n");
                 printf("\n\n\tDATA FILE = %s\n", pathname);
     printf("\n\tNNODES = %d\n\tNPORTS = %d",NNODES,NPORTS);
                printf("\n\n\n\n");
                break;
            } /* end switch */

printf("\nDO YOU WANT TO WRITE THE NEW DATA FILE ?  [y]\n");
             printf("\nHIT [enter] KEY TO BE CONTINUE\n ");
             getchar();
             switch (c = getchar())
                   {
                    case 'y':
                         printf("\n\n\n\n\n\n\n\n\n\n");
                         printf("\n\n\n\n\n\n\n\n\n\n");
                   printf("\n\n\tDATA FILE = %s\n", pathname);
      printf("\n\tNNODES = %d\n\tNPORTS = %d",NNODES,NPORTS);
                         printf("\n\n\n\n");
   printf("\n\tPROG.  WRITE DATA DSPM STRUCTURE FILE\n\n\n");
       printf("\n\tENTER NAME OF DATA FILE \"[_:_fn_._ft_]");
         printf("\n\t----> SPECIFY DISK DRIVE A:, B: OR C:");
        printf("\n\t----> AND FILENAME.DAT \(C:DSPM6.DAT\)");

                        do {
                       printf("\n\n\n\n ENTER FILE NAME = ");
                            scanf("%s",pathname);
                            printf("\n\n\n\n\n\n\n\n\n\n");
                            printf("\n\n\n\n\n\n\n\n\n\n");
                   printf("\n\n\tDATA FILE = %s\n", pathname);
      printf("\n\tDO YOU WANT TO CHANGE FILE NAME?  [y]\n");
             printf("\tHIT [enter] KEY TO BE CONTINUE\n\t");
                            getchar();
                            switch (c = getchar())
                                  {
                                   case 'y':
                   printf("\n\n\tDATA FILE = %s\n",pathname);
```

```
                        printf("\n\n\tWANT TO BE CHANGED");
                                printf("\n\n\n\n");
                                break;
                            default:
                                c = 'n';
                            printf("\n\n\n\n\n\n\n\n\n\n");
                            printf("\n\n\n\n\n\n\n\n\n\n");
                    printf("\n\n\tDATA FILE = %s\n",pathname);
      printf("\n\tNNODES = %d\n\tNPORTS = %d",NNODES,NPORTS);
                                printf("\n\n\n\n");
                                break;
                            } /* end switch */
                          } while (c == 'y');
                        /* write data file routine */
                        wrtfile();
                        break;
                    default:
                        printf("\n\n\n\n\n\n\n\n\n\n");
                        printf("\n\n\n\n\n\n\n\n\n\n");
                    printf("\n\n\tDATA FILE = %s\n",pathname);
printf("\n\tNNODES   =   %d\n\tNPORTS  =  %d",NNODES,NPORTS);
                        printf("\n\n\n\n");

                        /* go back to menu program */
                        menu();
                        break;
                    } /* end switch */

            /* go back to menu program */
            menu();
            break;
        case '3':

/* subroutine read data NODEnCNN[][] from dspm#.dat file */
            rdfile();

            /* error checking routine */
            checkncn();

        default:
            printf("\n----> EXIT FROM MENU <--------");
            printf("\nHIT [enter] KEY TO BE CONTINUE\n ");
            getchar();
            break;
      } /* end switch */
} /* end menu */
/*************************************************************/
/*                    END  MENU()  MODULE                    */
/*************************************************************/
```

APPENDIX C

CRPL() MODULE

```
/*************************************************************/
/*                                                           */
/*    PROGRAM   SUBROUTINE CREATE DATA PORT CONNECTION       */
/*              AND LINK STATUS FROM NODE_CNN[][] ARRAY      */
/*    FILE NAME SUB_CRPL.C  [CRPL()]                         */
/*                                                           */
/*************************************************************/

/* PROGRAM REQUIRES 2 ROUTINES:
   PRPCN()     (port connection display routine)
   PRNLK()     (link status display routine) */

#include <stdio.h>

#define  NOT_CONN -1
#define  USELESS   2
#define  BUS_CONT  Ø
#define  NUMNODES  NNODES+1
#define  NUMPORTS  NPORTS+1

/* define connection array 2 dimensions */
   int   NODE_CNN [33][7];
   int   PORT_CNN [33][7];
/* define status array 2 dimensions */
   int   LINK_STATUS [33][7];
/* define variables */
   int   NNODES,NPORTS;
   char  *pathname[15];

crpl()
    {
     extern int  NODE_CNN[][7];
     extern int  PORT_CNN[][7];
     extern int  LINK_STATUS[][7];
     extern int  NNODES,NPORTS;
     extern char *pathname[15];

     /* define variables */
     int ncn_count[33]; /* node count */
     int NODE,PORT;
     int i,z;
```

```
    for (NODE = 0; NODE < NUMNODES; NODE++)
       ncn_count[NODE] = 0; /* initialize node counter */
    for (NODE = 0; NODE < NUMNODES; NODE++)
       for (PORT = 0; PORT < NUMPORTS; PORT++)
          {
           z = NODE_CNN[NODE][PORT];
           if (z == NOT_CONN)
             {
              PORT_CNN[NODE][PORT] = z;
              /* set port connection  = -1 */
              LINK_STATUS[NODE][PORT] = USELESS;
               /* set link status to USELESS status */
             }
           if (z == BUS_CONT)
             LINK_STATUS[NODE][PORT] = USELESS;
               /* set link status to USELESS status */

           if (z != NOT_CONN)
             for (i = 0; i < NUMPORTS; i++)
                if (NODE_CNN[z][i] == NODE)
                  PORT_CNN[z][i] = PORT;
          } /* end for port & node */

    for (NODE = 0; NODE < NUMNODES; NODE++)
       if (ncn_count[NODE] > NUMNODES)
         printf("\nerror nodecount = %d",ncn_count[NODE]);

    /* port connection display routine */
    prpcn(NUMNODES,NUMPORTS);

    printf("\n\n\tHIT [enter] KEY TO BE CONTINUE\n\t");
    getchar();

    /* link status display routine */
    prnlk(NUMNODES,NUMPORTS);

    printf("\n\n\tHIT [enter] KEY TO BE CONTINUE\n\t");
    getchar();

   }  /* end crpl() */

/*************************************************************/
/*                 END  CRPL()  MODULE                       */
/*************************************************************/
```

APPENDIX D

GROWTH() MODULE

Growth() module is designed to prove the correctness of DSPM data structure output of growth algorithm, and can be tested performance for the specific configure() module which determines time performance of communication protocol in DSPM network. This module is a specific for interactive output data structure in analytical simulation. Which is the addition version of growth process in figure 16.

```
/*************************************************************/
/*                                                           */
/*   PROGRAM   SUBROUTINE GROWTH FOR DSPM SIMULATION         */
/*   FILE NAME SUB_GRWT.C [GROWTH()]                         */
/*                                                           */
/*************************************************************/

/* PROGRAM REQUIRES 1 ROUTINE:
   PRPST() (port status display routine) */

                #include <stdio.h>

                #define  INACTIVE     0
                #define  INBOUND      1
                #define  OUTBOUND    -1
                #define  USELESS      2
                #define  FAILED       1
                #define  MEMBER       1
                #define  NOT_CONN    -1
                #define  NOT_A_MEMBER 0
                #define  BUS_CONT     0
                #define  NUMNODES NNODES+1
                #define  NUMPORTS NPORTS+1

        /* DEFINE CONNECTION ARRAY 2 DIMENSIONS */
                int      NODE_CNN [33] [7];
                int      PORT_CNN [33] [7];
```

```
        /* DEFINE STATUS ARRAY 2 DIMENSIONS */
                int     PORT_STATUS [33] [7];
                int     LINK_STATUS [33] [7];

        /* DEFINE NETWORK TABLE ARRAY 1 DIMENSIONS */
                int     GROWTH_FIFO[33];
                int     NODE_ROOT  [33];
                int     NET_MEMBER [33];
                int     INPORT     [33];

        /* DEFINE VARIABLES INPUT INFORMATION*/
                int     NNODES,NPORTS;
                CHAR    *PATHNAME  [15];

      growth()
              {
               extern int NNODES,NPORTS;
               extern int NODE_CNN    [][7];
               extern int PORT_CNN    [][7];
               extern int PORT_STATUS[][7];
               extern int LINK_STATUS[][7];
               extern int GROWTH_FIFO[33];
               extern int NODE_ROOT   [33];
               extern int NET_MEMBER  [33];
               extern int INPORT      [33];

               /* DEFINE VARIABLES */
                     int INSERTION_POINT;
                     int EXTRACTION_POINT;
                     int GROWTH_POINT;
                     int NEXT_NODE;
                     int NODE, PORT;
                     int INIT = -9;
                     int i,j;

/*  Initialize pointer from data structures   */

      for ( NODE = 0; NODE <= NNODES; NODE++ )
         {
          NODE_ROOT[NODE]    = INIT;
          GROWTH_FIFO[NODE] = INIT;
          NET_MEMBER[NODE]   = NOT_A_MEMBER;
          for ( PORT = 0; PORT <= NPORTS; PORT++ )
               PORT_STATUS[NODE][PORT] = INACTIVE;
               /* next port */
         }  /* end for node */
            /* initialize pointer */
      INSERTION_POINT  = 0;
      EXTRACTION_POINT = 0;
      for ( PORT = 0; PORT <= NPORTS; PORT++ )
         {
```

```
            if ( LINK_STATUS[BUS_CONT][PORT] != USELESS )
            PORT_STATUS[BUS_CONT][PORT] = OUTBOUND;
          }  /* end for port */
        GROWTH_POINT = BUS_CONT;
        /* initialize GROWTH_POINT = BUS_CONT */
        GROWTH_FIFO[INSERTION_POINT] = 0;
        INSERTION_POINT += 1;
        /* increment insertion point */


/*************************************************************/
/*                    GROWTH  PROCESS                        */
/*************************************************************/

for ( NODE = 0; NODE <= NNODES; NODE++ )
   {
    for ( PORT = 0; PORT <= NPORTS; PORT++ )
       {
        if ( LINK_STATUS[GROWTH_POINT][PORT] != USELESS )
          {
           NEXT_NODE = NODE_CNN[GROWTH_POINT][PORT];
           if ( NET_MEMBER[NEXT_NODE] == NOT_A_MEMBER )
             {
              if ( GROWTH_POINT == BUS_CONT )
                 NODE_ROOT[NEXT_NODE] = PORT;
              else
                {
                 NODE_ROOT[NEXT_NODE] =
                 NODE_ROOT[GROWTH_POINT];
                 PORT_STATUS[GROWTH_POINT][PORT] = OUTBOUND;
                } /*end if growth_point = bus_cont */
              if ( CONFIGURE(GROWTH_POINT,NEXT_NODE) )
                {
                 INPORT[NEXT_NODE] =
                 PORT_CNN[GROWTH_POINT][PORT];
                 NET_MEMBER[NEXT_NODE] = MEMBER;
                 GROWTH_FIFO[INSERTION_POINT] = NEXT_NODE;
                 INSERTION_POINT += 1;
                } /* end configure */
             }  /* end if not a member */
          }  /* else if NEQ useless */
        else if ((NODE_CNN[GROWTH_POINT][PORT] == 0) &
                   (PORT_STATUS[BUS_CONT][(PORT_CNN
                    [GROWTH_POINT][PORT]) == OUTBOUND))
                PORT_STATUS[GROWTH_POINT][PORT] = INBOUND;
          /* end if NEQ useless */
       }  /* end for  port */

    GROWTH_POINT = GROWTH_FIFO[EXTRACTION_POINT];
    EXTRACTION_POINT += 1;
```

```
    printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
    printf("\t\t  VARIABLES ARRAY\n\n");
    printf("NODE");
    printf("\tNET_MEMBER\tINPORT\t   GROWTH_FIFO\n\n");

    for (j=0; j<NUMNODES; j++)
       {
        printf("  %d\t    %d\t",j,NET_MEMBER[j]);
        printf("\t   %d\t",INPORT[j]);
        printf("\t %d\n",GROWTH_FIFO[j]);
       } /* end for */

    printf("\n\n\nHIT [enter] KEY TO CONTINUE\n ");
    getchar();

   }  /* end for node */

/* port status display routine */
prpst(NUMNODES,NUMPORTS);

} /* end growth() */


/*************************************************************/
/*                END  GROWTH()  MODULE                      */
/*************************************************************/
```

APPENDIX E

INWRTFN() MODULE

```
/*************************************************************/
/*                                                           */
/*    PROGRAM   SUBROUTINE INPUT DATA NODE_CNN[][] FROM      */
/*              KEYBOARD AND CREATE DATA FILE DSPM#.DAT      */
/*    FILE NAME SUB_INWR.C  [INWRTFN()]                      */
/*                                                           */
/*************************************************************/

/* PROGRAM REQUIRES 3 ROUTINES:
   INPUTINFO()     (input file name, NNODES and NPORTS)
   INPUTNCN()      (input from keyboard for NODE_CNN[][])
   WRTFILE()       (create dspm#.dat data file) /*

#include  <stdio.h>

#define   NOT_CONN -1
#define   NUMNODES NNODES+1
#define   NUMPORTS NPORTS+1
/* define external parameters */
int  NODE_CNN[][7];
int  NNODES,NPORTS;
char *pathname[15];

inwrtfn()
      {
/* display information for input node_cnn selection menu */
        printf("\n\n\n\n\n\n\n\n\n");
        printf("\n\n\n\n\n\n\n\n\n");
        printf("\n\tPROG.  INPUT DATA DSPM STRUCTURE AND
                WRITE DSPM#.DAT FILE\n\n\n");
        printf("\n\n");
        printf("\n\tENTER NAME OF DATA FILE \"[_:_fn_._ft_]\"
                \(c:dspm6.dat\)");
        printf("\n\tENTER # OF NODES Ø TO NNODES FOR 1 <=
        NNODES <= 32");
        printf("\n\tENTER # OF PORTS Ø TO NPORTS FOR 1 <=
        NPORTS <= 6");

/* input filename & #NODES,#PORTS for NODE_CNN[][] routine */
        inputinfo();

/* input NODE_CNN[][]  routine */
        inputncn();
```

```
        /* write dspm data file routine */
        wrtfile();

        printf("\nHIT [enter] KEY TO CONTINUE\n ");
        getchar();
       } /* end inwrtfn */


/* END INWRTFN() SUBMODULE */


/* INPUT FILENAME & NNODES,NPORTS FOR NODE_CNN[][] ROUTINE */

inputinfo()
         {
          extern int NNODES,NPORTS;
          extern char *pathname[15];
          int c;
          do{
             printf("\n\n\n\n\n\n\n\n\n\n");
             printf("\n\n  ENTER NAME = ");
             scanf("%s",pathname);
             do{
                printf("\n\n  ENTER NNODES = ");
                scanf("%d",&NNODES);
                if ((NNODES < 1) || (NNODES > 32))
                  {
                   printf("\n ENTER NNODES 1 -> TO 32 ");
                   printf("\n  ----> ERROR <---- try again
                          \n");
                   } /* end if */
               } while ((NNODES < 1) || (NNODES > 16));
             do{
                printf("\n\n  ENTER NPORTS = ");
                scanf("%d",&NPORTS);
                if ((NPORTS < 1) || (NPORTS > 6))
                  {
                   printf("\n  ENTER NPORTS 1 -> TO 6 ");
                   printf("\n  ----> ERROR <---- try again
                          \n");
                   } /* end if */
               } while ((NPORTS < 1) || (NPORTS > 6));
             printf("\n\n\n\n\n\n\n\n\n\n");
             printf("\n\n\n\n\n\n\n\n\n\n");
             printf("\n\n\tDATA FILE = %s\n",pathname);
             printf("\n\tNNODES = %d\n\tNPORTS = %d",NNODES,
                    NPORTS);
             printf("\n\n\n\n");
             printf("ANY CHANGE ? [y]   HIT [enter] KEY TO
                    BE CONTINUE\n ");
             getchar();
```

```
            switch (c = getchar())
                  {
                   case 'y':
                        printf("\n\n\n\n\n\n\n\n\n\n");
                        printf("\n\n\tDATA FILE = %s\n",
                                pathname);
                        printf("\n\tNNODES = %d\n\tNPORTS =
                                %d",NNODES,NPORTS);
                        printf("\n\n\tTO BE CHANGED");
                        break;
                   default:
                        c = 'n';
                        printf("\n\n\n\n\n\n\n\n\n\n");
                        printf("\n\n\n\n\n\n\n\n\n\n");
                        printf("\n\n\tDATA FILE = %s\n",
                                pathname);
                        printf("\n\tNNODES = %d\n\tNPORTS =
                                %d",NNODES,NPORTS);
                        printf("\n\n\n\n\n\n\n\n\n\n");
                        printf("\n\tENTER NODE CONNECTION
                          ARRAY[%d][%d]",NUMNODES,NUMPORTS);
                        printf("\n\n\n\n");
                        break;
                  } /* end switch */
            } while (c == 'y'); /* end do while */

         } /* end inputinfo */

/* END INPUTINFO() ROUTINE */

/* WRITE DSPM DATA STRUCTURE INPUT FILE */

wrtfile()
       {
        extern int NODE_CNN[][7];
        extern int NNODES,NPORTS;
        extern char *pathname[15];

/* define dspm structure */
   static struct dspm {
        char name[15];
        int  nnodes;
        int  nports;
        int  ncn[240]  /*  node_connection */
       } d,*ptd;

/* define variables */
        int NODE, PORT;
        int size,fih;
        int i;
        ptd = &d;
```

```
/* write dspm#.dat file  */
        strcpy(ptd->name,pathname);
        ptd->nnodes = NNODES;
        ptd->nports = NPORTS;
        i = 0;

        for ( NODE = 0; NODE < NUMNODES; NODE++ )
           for ( PORT = 0; PORT < NUMPORTS; PORT++ )
              ptd->ncn[i++] = NODE_CNN[NODE][PORT];
              size = sizeof(struct dspm);
              fih = open(pathname,O_CREAT|O_BINARY|O_RDWR|
                                  O_TRUNC,S_IWRITE);
              write(fih,ptd,size);
              close(fih);
       } /* end wrtfile */

/*************************************************************/
/*      END  INWRTFN()  ROUTINE                              */
/*************************************************************/
```

APPENDIX F

RDFILE() MODULE

```
/*************************************************************/
/*                                                           */
/*    PROGRAM   SUBROUTINE READ INPUT DATA FILE DSPM#.DAT    */
/*              FROM FLOPPYDISK OR HARDDISK                  */
/*    FILE NAME SUB_RDFN.C  [RDFILE()]                       */
/*                                                           */
/*************************************************************/

/* PROGRAM REQUIRES 1 ROUTINE:
   PRNCN() (display node connection array) */

#include <stdio.h>

#define  NUMNODES NNODES+1
#define  NUMPORTS NPORTS+1

/* define external parameters */
int  NODE_CNN[][7];
int  NNODES,NPORTS;
char *pathname[15];

/*  read data file dspm#.dat */

rdfile()
      {
       extern int NNODES,NPORTS;
       extern int NODE_CNN[][7];
       extern char *pathname[15];

       int c;

/* enter file name for reading data input */
       printf("\n\n\n\n\n\n\n\n\n");
       printf("\n\n\n\n\n\n\n\n\n");
   do{
      printf("\n\n");
      printf("\n\tPROG.  INPUT DATA DSPM STRUCTURE FILE
               \n\n\n");
      printf("\n\tENTER NAME OF DATA FILE \"[_:_fn_._ft_]");
      printf("\n\t----> SPECIFY DISK DRIVE A:, B: OR  C:");
      printf("\n\t----> AND FILENAME.DAT \(C:DSPM6.DAT\)");
```

```
            do{
               printf("\n\n\n\n  ENTER FILE NAME = ");
               scanf("%s",pathname);
               printf("\n\n\n\n\n\n\n\n\n\n");
               printf("\n\n\n\n\n\n\n\n\n\n");
               printf("\n\n\tDATA FILE = %s\n",pathname);
               printf("\n\tDO YOU WANT TO CHANGE FILE NAME? [y]
                       \n");

               printf("\tHIT [enter] KEY TO CONTINUE\n\t");
               getchar();
               switch (c = getchar())
                     {
                      case 'y':
                           printf("\n\n\tDATA FILE = %s\n",
                                   pathname);
                           printf("\n\n\tWANT TO BE CHANGED");
                           printf("\n\n\n\n");
                           break;

                      default:
                           c = 'n';
                           printf("\n\n\n\n\n\n\n\n\n\n");
                           printf("\n\n\n\n\n\n\n\n\n\n");
                           printf("\n\n\tDATA FILE = %s\n",
                                   pathname);
                           printf("\n\n\n\n");
                           break;

              } while (c == 'y');

              /* read input data file routine */
              rd_ncn();

            } while (NNODES == 0);

           /* node connection  display  routine */
           prncn(NUMNODES,NUMPORTS);

           printf("\n\n\tDATA FILE = %s\n",pathname);
           printf("\n\tNNODES = %d\n\tNPORTS = %d",NNODES,
                   NPORTS);

           printf("\n\tHIT [enter] KEY TO CONTINUE\n ");
           getchar();

         } /* end rdfile */

/* END RDFILE() MODULE */
```

```
/* READ DSPM INPUT DATA FILE ROUTINE */

rd_ncn()
      {
       extern int NODE_CNN[][7];
       extern int NNODES,NPORTS;

       /* define structure */
       static struct dspm {
            char name[15];
            int  nnodes;
            int  nports;
            int  ncn[240]  /*  node_connection */
          } d,*ptd;

       /* define variables */
       int NODE, PORT;
       int size,fih;
       int i,j;
       ptd = &d;
       size = sizeof(struct dspm);
       fih = open(pathname,O_RDONLY|O_BINARY);
       if (fih == -1)
         {
          printf("\n\tNO FILE %s IN DISK SPECIFIED ",
                  pathname);
          printf("\n\t************> TRY AGAIN <***********");
          NNODES = 0;
         }
        else
         {
          read(fih,ptd,size);
          close(fih);
          ptd = &d;
          strcpy(pathname,ptd->name);
          NNODES = ptd->nnodes;
          NPORTS = ptd->nports;
          i = 0;
          for ( NODE = 0; NODE < NUMNODES; NODE++ )
             for ( PORT = 0; PORT < NUMPORTS; PORT++ )
                NODE_CNN[NODE][PORT] = ptd->ncn[i++];
         } /* end if */
      } /* end rd_ncn() */

/* END READ DATA FILE DSPM#.DAT ROUTINE */

/*************************************************************/
/*              END  RDFILE()  MODULE                        */
/*************************************************************/
```

APPENDIX G

INPUTNCN() ROUTINE

```
/*************************************************************/
/*                                                           */
/*     PROGRAM   SUBROUTINE INPUT NODE CONNECTION            */
/*               NODE_CNN[][] FROM KEYBOARD AND CHANGING     */
/*               OR MODIFYING DATA STRUCTURE                 */
/*     FILE NAME RT_INPUT.C  [INPUTNCN() ROUTINE]            */
/*                                                           */
/*************************************************************/

/* PROGRAM REQUIRES 3 ROUTINES:
   PRNCN()          (display output node_cnn[][])
   CHECKNCN()       (check input data error for NODE_CNN[][])
   CHANGEDATA()     (changing and or modifying data in
                     NODE_CNN[][] array) */

#include <stdio.h>

#define NOT_CONN -1
#define NUMNODES NNODES+1
#define NUMPORTS NPORTS+1

/* define external parameters */
int  NODE_CNN[33][7];
int  NNODES,NPORTS;
char *pathname[15];

inputncn()
        {
         extern int NNODES,NPORTS;
         extern int NODE_CNN[][7];
         extern char *pathname[15];
         int NODE, PORT;
         int c,n,p,z,*pti;

/* input NODE_CNN[NODE][PORT] */
for (NODE = 0; NODE < NUMNODES; NODE++)
   {
    for (PORT = 0; PORT < NUMPORTS; PORT++)
       {
        do{
           printf("\n%s%d%s%d%s","ENTER NODE_CNN[",NODE,",",
                   PORT,"] = ");
           scanf("%d",pti);
```

```
        z = *pti;
        if ((z < NOT_CONN) || (z >= NUMNODES))
          {
           printf("\nENTER NODE_CNN -1 -> TO %d ",NNODES);
           printf("\n----> ERROR <---- try again \n");
          } /* end if */
      } while ((z < NOT_CONN) || (z >= NUMNODES));
     NODE_CNN[NODE][PORT] = z;
     /* set node connection to connect to node z*/
    } /* end for port */
if ((NODE+1) < NUMNODES)
  {
   /* node connection display routine */
   prncn(NODE+1,NUMPORTS);
   printf("\n\n");
   do{
      printf("\n\nANY CHANGE IN NODE[Ø->%d] PORT[Ø->%d] ?
              [y]\n",NODE,NPORTS);
      printf("\nHIT [enter] KEY TO BE CONTINUE\n ");
      getchar();
      switch ( c = getchar())
            {
             case 'y':
                  do{
                     printf("ENTER NODE,PORT,NODE_CNN");
                     printf("\n\nENTER NODE = ");
                     scanf("%d",&n);
                     printf("\nENTER PORT = ");
                     scanf("%d",&p);
                     if (((n < Ø) || (n > NODE)) ||
                         ((p < Ø) || (p > NPORTS)))
                       {
                        n = -1;
                        printf("\n---->ERROR<---- try
                                again \n");
                        printf("\n\n\n\n");
                       } /* end if */
                   } while (n == -1);
                  do{
                     printf("\n%s%d%s%d%s","ENTER
                             NODE_CNN[",n,",",p,"] = ");
                     scanf("%d",pti);
                     z = *pti;
                     if ((z < NOT_CONN) || (z >=
                          NUMNODES))
                       {
```

```
                                printf("\nENTER NODE_CNN -1 -> TO
                                        %d ",NNODES);
                                printf("\n----> ERROR <---- try
                                        again \n");
                                } /* end if */
                              } while ((z < NOT_CONN) || (z >=
                                        NUMNODES));
                               NODE_CNN[n][p] = z;
                              /* node connection display routine */
                               prncn(NODE+1,NUMPORTS);
                             break;
                    default:
                             c = 'n';
                             printf("\n\n\n\n\n\n\n\n\n\n");
                             printf("\n\n\n\n\n\n\n\n\n\n");
                             /* node connection display routine */
                             prncn(NODE+1,NUMPORTS);
                             printf("\n\n\n\n");
                             break;
                   } /* end switch */
          } while (c == 'y'); /* end do while */
       } /* end if node */
    } /* end for node */

/* node connection display routine */
prncn(NUMNODES,NUMPORTS);

/* error checking routine */
checkncn();

printf("\n\n\n\n");
printf("DO YOU WANT ANY CHANGE ?\n");
printf("\n*--->ENTER  [a] TO CHANGE ALL AT THE THE BEGINNING"
      );
printf("\n             [b] TO CHANGE AT EACH NODE_CNN[][]\n");
printf("\nHIT[enter] key OR ANY CHARACTER TO BE CONTINUE\n");
getchar();
switch (c = getchar())
       {
        case 'a':

             /* input NODE_CNN[][] routine */
             inputncn();
             break;
        case 'b':
```

```
            /* changing data structure routine */
            changedata();
            break;
        default:

            /* node connection display routine */
            prncn(NUMNODES,NUMPORTS);

           /* error checking routine */
            checkncn();

            printf("\n\n\tDATA FILE = %s\n",pathname);
            printf("\n\tNNODES = %d\n\tNPORTS = %d\n",NNODES,
                    NPORTS);
            break;
      } /* end switch */
} /* end inputncn() */


/*************************************************************/
/*        END  INPUTNCN()  ROUTINE                           */
/*************************************************************/
```

APPENDIX H

CHANGEDATA() ROUTINE

```
/*************************************************************/
/*                                                           */
/*    PROGRAM   ROUTINE CHANGING DATA NODE CONNECTION        */
/*    FILE NAME RT_CHGNC.C  [CHANGEDATA()]                   */
/*                                                           */
/*************************************************************/

/* PROGRAM REQUIRES 2 ROUTINES:
   PRNCN()      (node connection display routine)
   CHECKNCN()   (check error data node connection) */

#include <stdio.h>

#define  NUMNODES  NNODES+1
#define  NUMPORTS  NPORTS+1

/* define connection array 2 dimensions */
   int   NODE_CNN [33][7];
/* define variables */
   int   NNODES,NPORTS;
   char  *pathname[15];

changedata()
          {
           extern int  NODE_CNN[][7];
           extern int  NNODES,NPORTS;
           extern char *pathname[15];
           int *pti;

           /* define variables */
           int *pti;
           int c,i,j,n,p,z;

/* node connection display routine */
prncn(NUMNODES,NUMPORTS);

printf("\n\n\tDATA FILE = %s\n",pathname);
printf("\n\tNNODES = %d\n\tNPORTS = %d",NNODES,NPORTS);
printf("\n\n");
printf("CHANGING IN NODE[0->%d] PORT[0->%d] ? [y]\n",NNODES,
       NPORTS);
printf("\nENTER NODE,PORT,NODE_CNN\n");
```

```
do {
    printf("\n\n  ENTER NODE = ");
    scanf("%d",&n);
    if ((n < 0) || (n > NNODES))
      {
       printf("\nENTER NODE 1 -> TO %d",NNODES);
       printf("\n----> ERROR <---- try again \n");
      } /* end if */
   } while ((n < 0) || (n > NNODES));
do {
    printf("\n\n  ENTER PORT = ");
    scanf("%d",&p);
    if ((p < 0) || (p > NPORTS))
      {
       printf("\nENTER PORT 1 -> TO %d",NPORTS);
       printf("\n----> ERROR <---- try again \n");
      } /* end if */
   } while ((p < 0) || (p > NPORTS));
do{
   printf("\n%s%d%s%d%s","ENTER NODE_CNN[",n,",",p,"] = ");
   scanf("%d",pti);
   z = *pti;
   if ((z < -1) || (z > NNODES))
     {
      printf("\nENTER NODE_CNN -1 -> TO %d ",NNODES);
      printf("\n----> ERROR <---- try again \n");
     } /* end if */
   NODE_CNN[n][p] = z;
  } while ((z < -1) || (z >= NUMNODES));

/* error checking routine */
checkncn();

printf("\n\nANY CHANGE IN NODE[0->%d] PORT[0->%d] ? [y]\n",
        NNODES,NPORTS);
printf("\nENTER *--> n OR ANY CHARACTER TO BE CONTINUE\n");
z = getchar();
c = z;
while ((c == '\n') || ( c == EOF))
     c = getchar();
switch (c)
      {
       case 'y':

       /* changing data structure routine */
            changedata();
```

```
        default:
        /* error checking routine */
             checkncn();

             printf("\nHIT [enter] KEY TO BE CONTINUE\n ");
             getchar();
             break;
        } /* end switch */

} /* end changedata */

/*************************************************************/
/*          END CHANGEDATA() ROUTINE                         */
/*************************************************************/
```

APPENDIX I

CHECKNCN() ROUTINE

```
/*************************************************************/
/*                                                           */
/*     PROGRAM   ROUTINE CHECK ERROR DATA NODE CONNECTION    */
/*     FILE NAME RT_CKNCN.C  [CHECKNCN()]                    */
/*                                                           */
/*************************************************************/


#include <stdio.h>

#define  NUMNODES  NNODES+1
#define  NUMPORTS  NPORTS+1

/* define connection array 2 dimensions */
   int   NODE_CNN [33][7];
/* define variables */
   int   NNODES,NPORTS;

checkncn()
        {
         extern int  NODE_CNN[][7];
         extern int  NNODES,NPORTS;

         /* define variables */
         static int ncn_count[33];
         static int ncn_err[33];
         int NODE,PORT;
         int i,j,z;


/* CHECK NODE CONNECTION ARRAY[NODE][PORT] */

for (NODE = 0; NODE < NUMNODES; NODE++)
   {
    ncn_count[NODE] = 0;
    /* initialize node counter connection */
    ncn_err[NODE] = 0;
   } /* end for node */
for (NODE = 0; NODE < NUMNODES; NODE++)
   for (PORT = 0; PORT < NUMPORTS; PORT++)
      {
       z = NODE_CNN[NODE][PORT];
       j = 0;
       for (i = 0; i < NUMPORTS; i++)
```

```
          {
            if (z == -1)
              break;
            if (NODE_CNN[z][i] == NODE)
              {
               j += 1;
               ncn_err[z] = j;
              } /* end if */
           } /* end for i */

        if (z != -1)
          if ((ncn_err[z] != 1))
            {
             printf("\n--->NODE[%d] ERROR CONNECTION<------*",
                    z);
             printf("\n\n....NODE_CNN[%d][%d]-*/*->NODE[%d]...
                    \n",NODE,PORT,z);
            } /* end if */

        if ((z < -1) || (z > NNODES))
           printf("\nNODE_CNN[%d][%d] = %d <-----ERROR
                  CONNECTION",NODE,PORT,z);
        ncn_err[z] = 0;
        ncn_count[z] += 1;
       } /* end for */

for ( i = 0; i < NUMNODES; i++)
    if (ncn_count[i] > NUMPORTS)
      {
       printf("\nNODE[%d] ERROR \(excess\) CONNECTION",i);
       printf("\nNODE[%d] COUNT = %d  --> > #PORTS",i,
              ncn_count[i]);
      } /* end if */

} /* end checkncn */

/*************************************************************/
/*          END   CHECKNCN()  ROUTINE                        */
/*************************************************************/
```

APPENDIX J

DISPLAY OUTPUT ROUTINES

```
/*************************************************************/
/*                                                           */
/*    PROGRAM   DISPLAY OUTPUT ROUTINES                      */
/*    FILE NAME RT_ROUTS.C .                                 */
/*                                                           */
/*************************************************************/

/* DISPLAY OUTPUT ROUTINES ARE :
   PRNCN()      (node connection display routine)
   PRPCN()      (port connection display routine)
   PRPST()      (port status display routine)
   PRLKS()      (link status display routine) */

#include <stdio.h>

/* define connection array 2 dimensions */
   int   NODE_CNN [33][7];
   int   PORT_CNN [33][7];

/* define status array 2 dimensions */
   int   PORT_STATUS [33][7];
   int   LINK_STATUS [33][7];

/* NODE CONNECTION DISPLAY ROUTINE */

prncn(n,p)
int n,p;
     {
      extern int  NODE_CNN[][7];
      int i,j,z;

      printf("\n\n\n\n\n\n\n\n\n\n");
      printf("\n\n\n\n\n\n\n\n\n\n");
      printf("\n\n\n\n");
      printf("\t\t\tNODE CONNECTION\n\n");
      printf("\t\t\tPORT\n\t\t");

      for (i = 0; i < p; i++)
         printf("\t%d",i);

      printf("\n\tNODE\n");
```

```
      for (i = 0; i < n; i++)
         {
          printf("\t  %d\t",i);
          for (j = 0; j < p; j++)
             {
              z = NODE_CNN[i][j];
              if ((z < 0) || (z > 9))
                printf("\t\b%d",z);
              else
                printf("\t%d",z);
             } /* end for j */
          printf("\n");
         } /* end for i */
      } /* end prncn */

/* END NODE CONNECTION DISPLAY */


/* PORT CONNECTION DISPLAY ROUTINE */

prpcn(n,p)
int n,p;
     {
      extern int  PORT_CNN[][7];
      int i,j,z;

      printf("\n\n\n\n\n\n\n\n\n\n");
      printf("\n\n\n\n\n\n\n\n\n\n");
      printf("\n\n\n\n");
      printf("\t\t\tPORT CONNECTION\n\n");
      printf("\t\t\tPORT\n\t\t");

      for (i = 0; i < p; i++)
         printf("\t%d",i);
      printf("\n\tNODE\n");
      for (i = 0; i < n; i++)
         {
          printf("\t  %d\t",i);
          for (j = 0; j < p; j++)
             {
              z = PORT_CNN[i][j];
              if ((z < 0) || (z > 9))
                printf("\t\b%d",z);
              else
                printf("\t%d",z);
             } /* end for j */
          printf("\n");
```

```
         } /* end for i */
     } /* end prpcn */

/* END PORT CONNECTION DISPLAY */


/* PORT STATUS DISPLAY ROUTINE */

prpst(n,p)
int n,p;
     {
      extern int PORT_STATUS[][7];
      int i,j,z;

      printf("\n\n\n\n\n\n\n\n\n");
      printf("\n\n\n\n\n\n\n\n\n");
      printf("\n\n\n\n");
      printf("\t\t\tPORT STATUS\n\n");
      printf("\t\t\tPORT\n\t\t");

      for (i = 0; i < p; i++)
         printf("\t%d",i);
      printf("\n\tNODE\n");
      for (i = 0; i < n; i++)
         {
          printf("\t  %d\t",i);
          for (j = 0; j < p; j++)
             {
              z = PORT_STATUS[i][j];
              if ((z < 0) || (z > 9))
                printf("\t\b%d",z);
              else
                printf("\t%d",z);
             } /* end for j */
          printf("\n");
         } /* end for i */

      printf("\nHIT [enter] KEY TO CONTINUE\n ");
      getchar();

     } /* end prpst */

/* END PORT STATUS DISPLAY ROUTINE */
```

```
/* LINK STATUS DISPLAY ROUTINE */

prnlk(n,p)
int n,p;
     {
      extern int LINK_STATUS[][7];
      int i,j,z;

      printf("\n\n\n\n\n\n\n\n\n\n");
      printf("\n\n\n\n\n\n\n\n\n\n");
      printf("\n\n\n\n");
      printf("\t\t\tLINK STATUS\n\n");
      printf("\t\t\tPORT\n\t\t");

      for (i = 0; i < p; i++)
         printf("\t%d",i);
      printf("\n\tNODE\n");
      for (i = 0; i < n; i++)
         {
          printf("\t  %d\t",i);
          for (j = 0; j < p; j++)
             {
              z = LINK_STATUS[i][j];
              if ((z < 0) || (z > 9))
                printf("\t\b%d",z);
              else
                printf("\t%d",z);
             } /* end for j */
          printf("\n");
         } /* end for i */

     } /* end prnlk */

/* END LINK STATUS DISPLAY ROUTINE */

/*************************************************************/
/*           END  DISPLAY  ROUTINES                          */
/*************************************************************/
```

LIST OF REFERENCES

1. Johnson, Barry W., "Fault-Tolerant Microprocessor-Based Systems," IEEE Micro., Vol. 4, No. 6, December 1984.

2. Siewiorek, Daniel P., Bell, C. Gordon, and Allen, Newell, Computer Structures: Principles and Examples, McGraw-Hill, Inc., 1982.

3. Siewiorek, Daniel P., "Architecture of Fault-Tolerant Computers," IEEE Computer, Vol. 17, No. 8, August 1986.

4. Lala, Parag K., Fault Tolerant & Fault Testable Hardware Design, Prentice/Hall International, Inc., 1986.

5. Smith, Thomas B. III, "A Damage and Fault-Tolerant Input/Output Network," IEEE Trans, Computer, Vol. C-34, May 1975.

6. Abbott, Larry W., "Operational Characteristics of the Dispersed Sensor Processor Mesh," IEEE/AIAA Fifth Digital Avionics System Conference, Seattle, Washigton, October 31 - November 3, 1983.

7. Abbott, Larry W., "Test Experience on an Ultra-reliable Computer Communication Network," IEEE/AIAA Sixth Digital Avionics System Conference. Baltimore, Maryland, December 3-6, 1984.

8. Megna, Vincent A., "Tactical Airborne Distributed Computing and Networks," Dispersed Sensor Processing Mesh Project, AGARD-CCP-303, October 1981.

9. Garg, K., "An Approach to Performance Specification of Communication Protocols Using Timed Petri Nets," Proceeding The 4th International Conference on Distributed Computing Systems, San Francisco, California, May 14-18, 1984.

10. Tanenbaum, Andrew S., Computer Networks, Prentice-Hall, Inc., 1981.

11. Berthelot, Gerard, and Terrat, Richard, "Petri Nets Theory for the Correctness of Protocols," IEEE Transactions on Communications," Vol. Com-30, No. 12, December 1982.

12. Stallings, William, Data and Computer Communications, Macmillan, Inc., 1985.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 62 Electrical and Computer Engineering Dept. Naval Postgraduate School Monterey, California 93943-5000	1
4.	Professor Larry W. Abbott, Code 62Ab Electrical and Computer Engineering Dept. Naval Postgraduate School Monterey, California 93943-5000	2
5.	Professor Jon T. Butler, Code 62Bl Electrical and Computer Engineering Dept. Naval Postgraduate School Monterey, California 93943-5000	1
6.	Lt. Nophadol Sudhamasapa 33/8 Soi Maiyalap, Ramintra Rd. Bangapi, Bangkok Thailand	4

220859

Thesis
S858227
c.1

Sudhamasapa
A development and simulation of Synergistically Integrated Reliability (SIR) for an ultra-reliable fault tolerance computer under communication software protocol for the growth algorithm.